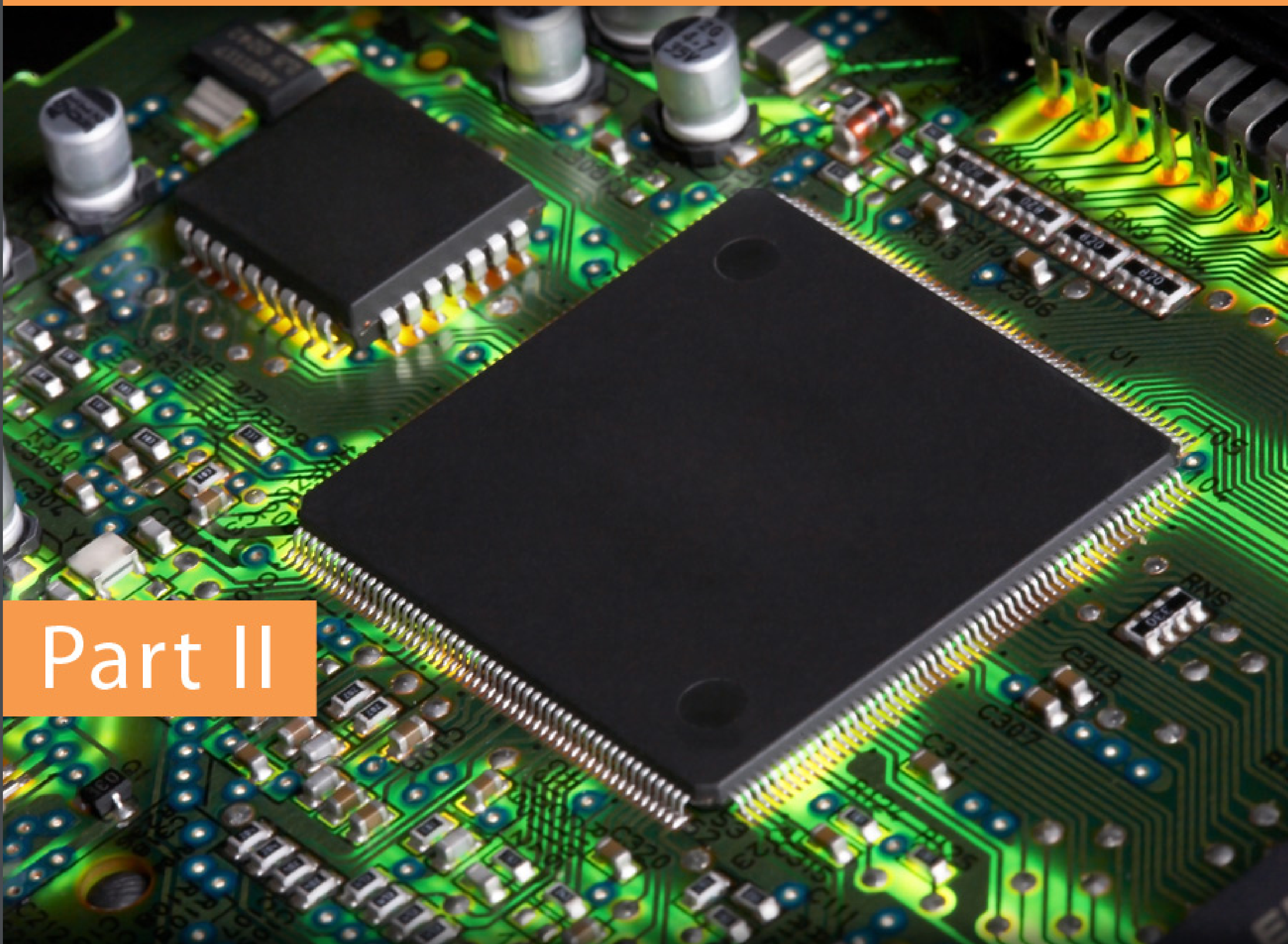
PaulOS: Part II

An 8051 Real-Time Operating System

Paul P. Debono



Part II

Paul P. Debono

PaulOS

An 8051 Real-Time Operating System

Part II

PaulOS: An 8051 Real-Time Operating System
Part II
1st edition

ISBN 978-87-403-0450-3

Contents

To see Part I download PaulOS Part I

12 Programming Tips and Pitfalls

In this final chapter we discuss some programming tips and common pitfalls which should be avoided when programming such micro-controllers.

12.1 RAM size

The 8051 may only address 64KB of RAM. To expand RAM beyond this limit requires programming and hardware tricks. We may have to do this "by hand" since many compilers and assemblers, while providing support for programs in excess of 64KB, do not support more than 64KB of RAM. This is rather strange since program code can usually fit in 64KB but it is often that data RAM that is lacking. Thus if we need more than 64KB of RAM, we need to check if our compiler supports it, but if it does not, we must be prepared to do it by hand.

Some assemblers and compilers offer ways to get around this limit when used with specially wired hardware. However, without such special compilers and hardware, program code is normally limited to 64KB for the standard 8051 micro-controller. Newer derivatives of the 8051, such as the Silicon Labs C8051F120 chip, do have 128KB of in-system programmable flash memory, with special SFRs to handle the extra RAM size. The latest software development tools, such as the KEIL IDE do provide methods for making use of this additional RAM, basically by switching in and out 64KB pages.

12.2 SP setting

If we only use the first register bank (i.e. bank 0), we may use Internal RAM locations 08h through 1Fh, for our own data use. However if we plan to use register banks 1, 2, or 3 we must be very careful about using addresses below 20h for our variables as we may end up overwriting or corrupting the values stored in our registers. In particular, the SP (used to point to the stack area) by default is loaded with 07 so that the stack starts from location 08. For example, if we are using Bank 1 together with Bank 0, we have to make sure to load SP with a higher value, such as 0Fh which is the address of R7 bank 1 (the highest register in use).

Similarly, if our program does not use any bit variables, then we may use Internal RAM locations 20h through 2Fh (Bit-addressable area) for our own use as normal data byte memory locations. On the other hand, if we intend to use some bit variables, we must be very careful as to which address we do initialize SP as once again we may end up overwriting the stored value of our bits whenever we push something on stack. As the stack grows upwards, it starts to over-write locations, starting from 08h. If there are a lot of pushes or calls, it might end up over-writing the bit variable area. Hence once again, the SP might need to be initially set to 2Fh if we need to preserve all the bit-addressable area.

12.3 SFRs

SFRs are used to control the way the 8051 peripherals functions. Not all the addresses above 80h are assigned to SFRs. However, this area may not be used as additional RAM memory even if a given address has not been assigned to an SFR. Free locations are reserved for future versions of the micro-controller and if we use that area, then our program would not be compatible with future versions of the microcontroller, since those same locations might be used for special additional SFRs in the upgraded version. Moreover, certain unused locations may actually be non-existent, in the sense that the actual cells for that memory would not form part of the memory mask when being manufactured, and hence even if we do write the code to use these locations, no actual data would be stored!

It is therefore recommended that we do not read from or write to any SFR addresses that have not been actually assigned to an SFR. Doing so may provoke undefined behaviour and may cause our program to be incompatible with other 8051 derivatives that use those free addresses to store the additional SFRs for some new timer or peripheral included in the new derivative.

If we write a program that utilizes the new SFRs that are specific to a given derivative chip (and which therefore were not included in the standard basic 8051 SFR list), our program will not run properly on a standard 8051 where those SFRs simply did not exist. Thus, it is best to use non-standard SFRs only if we are sure that our program will only have to run on that specific micro-controller. If we happen to write code that uses non-standard SFRs and subsequently share it with a third-party, we must make sure to let that party know that our code is using non-standard SFRs and can only be used with that particular device. Good remarks, notes and warnings within the program source listing would help.

12.4 Port usage

While the 8051 has four I/O ports (P0, P1, P2, and P3), if our hardware uses external RAM or external code memory (i.e. if our program is stored in an external ROM or EPROM chip or if we are using external RAM chips) we cannot use P0 or P2. This is because the 8051 uses ports P0 and P2 to address the external memory. Thus if we are using external RAM or code memory we may only use ports P1 (and perhaps P3 with some bit restrictions depending on the application program, since the P3 bits are also used as RD, WR, T1, T0, INT1, TXD and RXD) for our own use.

12.5 DPTR

DPTR is really a combination of two 8-bit registers DPH and DPL, taken together as a 16-bit value. In reality, we almost always have to deal with DPTR one byte at a time. For example, to push DPTR onto the stack we must first push DPL and then push DPH. We cannot simply push DPTR onto the stack as a 16-bit value in one step.

Additionally, there is an instruction to increment DPTR (which is INC DPTR). When this instruction is executed, the two bytes are operated upon as a 16-bit value. However, there is no assembly language instruction which decrements DPTR. If we wish to decrement the value of DPTR, we must write our own code to do so, such as:

```
CLR C
MOV A,DPL
SUBB A,#1
MOV DPL,A
MOV A,DPH
SUBB A,#0; subtract the carry flag from the first subtraction, if necessary
MOV DPH,A
```

12.6 Serial port (UART)

To use the 8051's on-board serial port, it is generally necessary to initialise at least the following four SFRs: SCON, PCON, TCON, and TMOD. This is because SCON on its own does not fully control the serial port. However, in most cases the program will need to use one of the timers to establish the serial port baud rate. In this case, it would be necessary to configure Timer 1 by setting TCON and TMOD. PCON.7 (known also as SMOD bit, but we should note that PCON is not a bit-addressable register), can be set to double the baud rate. In this case therefore, we would also need to program bit 7 of a fourth register PCON.

Moreover, if the serial handling routine is to run under interrupt control, then the appropriate interrupt enable bits (ES and EA in the IE SFR) and sometimes even the interrupt priority bit (PS in the IP SFR) have also to be set. This would bring to six the number of SFRs which we may need to set in order to use the UART in interrupt mode.

TI flag is normally initialized to 0 if using serial interrupt routines to transmit characters stored in some software buffer. Once SBUF is loaded directly with the first character to be transmitted, the transmission would start, with the start bit, bit 0 to bit 7 of the data, any parity bit, followed by the stop bit. TI would then be set to 1 automatically when this first character transmission is done and the ISR routine is then triggered which would continue to send any remaining characters in the software buffer (TI would need to be reset to 0 every time in the ISR code).

If however we are not using serial interrupt routines to transmit data, TI would be intialised to 1 in the first place, since it is usual practice to start the putchar() routine with:

```
while (TI==0);          // wait for the transmitter to be ready (TI=1)
SBUF = c;               // store character in SBUF and start transmitting character
                        // TI would be automatically set to 1 once transmission is done
```

Examples are given in the serial routines in the Appendix.

12.7 Interrupts

Forgetting to protect the PSW register: If we write an interrupt handler routine, it is a very good idea to always save the PSW SFR on the stack and restore it when our interrupt service routine (ISR) is complete. Many 8051 instructions modify the bits within PSW. If our ISR does not guarantee that PSW contains the same data upon exit as it had upon entry, then our program is bound to behave rather erratically and unpredictably. Moreover it will be tricky to debug since the behaviour will tend to vary depending on when and where in the execution of the program, the interrupt happened.

Forgetting to protect a Register: We must protect all our registers as explained above. If we forget to protect a register that we will use in the ISR and which might have been used in some other part of our program, very strange results may occur. If we are having problems with registers changing their value unexpectedly or having some arithmetic operations producing wrong answers, it is very likely that we have forgotten to protect some registers.

Forgetting to restore protected values: Another common error is to push registers onto the stack to protect them, and then we forget to pop them off the stack (or we pop them in the wrong order) before exiting the interrupt. For example, we may push ACC, B, and PSW onto the stack in order to protect them and subsequently pop only PSW and ACC off the stack before exiting. In this case, since the value of register B was not restored (popped), an extra value remains on the stack. When the RETI instruction is then executed at the end of the ISR, the 8051 will use that value as part of the return address instead of the correct value. In this case, the program will almost certainly crash. We must always ensure that the same number of registers are popped off the stack and in the right order:

```
PUSH PSW
PUSH ACC
PUSH B
…
…
…
POP B
POP ACC
POP PSW
RETI
```

Using the wrong register bank: Another common error occurs when calling another function or routine from within an ISR. Very often the called routine would have been written with a particular register bank in mind, and if the ISR is using another bank, there might be problems when referring to the registers in the called routine. If we are writing our own routine, then in the ISR we could save the PSW register, change the register bank and then restore the PSW register before exiting from the called routine. However, particularly if we are using the C compiler, we might be using functions and procedures pre-written in the compiler and which we do not have any control on, and therefore can result in program not functioning as intended.

This problem is particularly serious when using pre-emptive RTOSs (such as SanctOS or MagnOS), where a forced change of task might occur, switching from task A (which was using for example using register bank 1) on to task B which uses bank 2. For the case of co-operative RTOSs (such as PaulOS), we would be in control where the task changes occur and we would be able to take the necessary precautions.

Forgetting to re-start a timer: We might turn off a timer to re-load the timer register values or to read the counter in an interrupt service routine (ISR) and then forget to turn it on again before exiting from the ISR. In this case, the ISR would only execute once.

Forgetting to clear the Timer 2 interrupt flag: When using Timer 2 interrupts, the Timer 2 overflow flag TF2 is not cleared automatically when the ISR is serviced. We have to clear it in the ISR software (using CLR TF2). The same problem occurs if we forget to clear the RI or the TI flags when using the Serial Interrupt. In this case, the ISR keeps on being called repeatedly.

Using RET instead of RETI: Remember that interrupts are always terminated with the RETI instruction. It is easy to inadvertently use the RET instruction instead. However the RET instruction will not end our interrupt smoothly. Usually, using RET instead of RETI will cause the illusion of the main program running normally, but the interrupt will only be executed once. If it appears that the interrupt mysteriously stops executing, we must verify that RETI is being used.

Certain assemblers contain special features which will issue a warning if the programmer fails to protect registers or commit some other common interrupt-related errors.

12.8 RTOSs pitfalls

The PaulOS co-operative RTOS is the most robust and secure of the RTOSs which we have introduced in this text book. This is mainly due to the fact that being a co-operative RTOS, the task changes occur when we want them since there cannot be any forced pre-emptive task changes. However there can still be hidden problems. We should take special care when handling global variables which are accessible to all the tasks. We have to make sure that these variables are allowed to be manipulated only when we want them to. Otherwise it might happen that a task starts with one value of a global variable, then it goes on to a wait state, and when it later on resumes to run, it might end up using the wrong value of the same variable.

This is a very big problem with the SanctOS and MagnOS pre-emptive RTOSs. The safest way would be to have global variables protected as a resource, allowing them to be changed only when it is safe to do so. These pre-emptive RTOSs (SanctOS and MagnOS) are only written here as a proof of concept and not as a fully functional robust operating system. This has to be always kept in mind.

The same problem exists in these RTOSs with register banks and tasks which use the same functions which are non re-entrant.

12.8 C Tips

- We should always try to keep functions (or tasks) as simple as possible.
- Use the correct required types for the variables; do not use **int** type if we really need **byte** or **bit** type.
- Use signed or unsigned types correctly.
- Use specified locations for storing pointers. That is use declarations such as

```
char data * xdata str;          /* pointer stored in xdata, pointing to char stored in data */
int xdata * data numtab;        /* pointer stored in data, pointing to int stored in to xdata */
long code * idata powtab;       /* pointer stored in idata, pointing to long stored in code */
```

- In order to improve the performance during code execution or to reduce the memory size requirement for our code, we should analyse the generated list files and assembly code so as to determine which routines can be improved in speed or reduced in size.
- We should always try to minimize the variable usage by scoping.

Appendix A ParrOS.a51

Round-Robin RTOS

This is the round-robin real-time operating system version called ParrOS (an acronym for PAul's Round-Robin Operating System) and is perhaps the simplest operating system which can be written.

The operation can be explained as follows:

A timer interrupt is generated at regular intervals. This interrupt is used to run periodically a crucial Interrupt Service Routine (ISR). This ISR uses counters to determine accurately whether the specified slot time has passed, at which point a function is called which tackles the task-swapping problem. Mainly this function stores all the stack area for the current task and replaces it with the stack for the next task scheduled to run. At this point the jump is made to the new task and the program continues seamlessly with the new task until its slot time has elapsed. The process repeats indefinitely, looping round through all the tasks.

We first start by explaining how the variables are stacked in the internal memory area of the 8051. Table A-1 shows the way the variables used in this RTOS program have been set up. Most of the variables reside in the internal 256 RAM of the 8032 micro-processor. The external RAM (from address 8100H to 9FFFH for the Flight 32 board) is used to store the stacks of all the tasks and of the main idle program. These stacks are then swapped in turn with the area reserved for the stack in the internal RAM whenever a task swap is necessary.

Label	Hex Byte Addr.	Remarks Hex bit address								Notes
	FF To 80	Indirect General Purpose RAM (80 – FF) which can be used as a Stack Area								
MAIN_STACK	7F to 76	Direct and Indirect RAM (00 – 7F)								
SP (initially) T_SLOT_RELOAD	75 to 61	(NOOFTSKS+1) bytes								Time slot Reload values For each task
T_SLOT	60 to 4C	(NOOFTSKS+1) bytes								Time slot Counter For each task
SPTS	4B to 37	(NOOFTSKS+1) bytes								Storage area For the SPs Of each task
READYQ	35 to 22	(NOOFTSKS+1) bytes								Queue for Tasks ready To run
	21	0F	0E	0D	0C	0B	0A	09	08	Spare bits
	20	07	06	05	04	03	02	01	00	MYBITS
	1F to 17									Storage for any Applications variables
TMPSTORE0	16									See FETCH_STACK
GOPARAM	15									See RTOSGOXXX
DELAYHI	14									See RTOSGOXXX
DELAYLO	13									See RTOSGOXXX
TICKCOUNT	12									See RTOSGOXXX
RUNNING	11	Currently running task								Task number
READYQTOP	10	Points to last task in READYQ								Pointer
	0F to 08	Register Bank 1 (R0 – R7)								Register bank Used by the RTOS
	07 to 00	Register Bank 0 (R0 – R7)								Register bank used by ALL tasks

Table A-1 PARROS.A51 Variables setup, with 20 tasks. (NOOFTSKS=20)

The source listing for the ParrOS A51 program consists of:

- The header file ParrOS.h
- The startup file ParrOS_Startup.a51
- The main RTOS file ParrOS.a51

ParrOS.h

```
/* ParrOS.h */
/* for use with Parros.a51 Round-Robin RTOS program */
/* written by Paul P. Debono - November 2002 */

#define uchar unsigned char
#define uint unsigned int

// The following receive parameters, hence are declared
// with an underscore prefix in the a51 file

void INIT_RTOS(uchar tslot);
void RTOSGOMSEC(uchar msec);
void RTOSGOSEC(uchar sec);
void RTOSGOMIN(uchar min);
void CREATE(uchar task,uchar tslot,uint *taskadd);
```

==

ParrOS_StartUp.a51

```
$NOMOD51
;------------------------------------------------------------------------------
;  This file is part of the C51 Compiler package
;  Copyright (c) 1988-2005 Keil Elektronik GmbH and Keil Software, Inc.
;  Version 8.01
;
;  *** <<< Use Configuration Wizard in Context Menu >>> ***
;------------------------------------------------------------------------------
;  STARTUP.A51: This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;       A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the following
;  Lx51 invocation:
;
;       Lx51 your object file list, STARTUP.OBJ controls
;
;------------------------------------------------------------------------------
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;  IDATALEN: IDATA memory size <0x0-0x100>
;       Note: The absolute start-address of IDATA memory is always 0
```

```
;               The IDATA space overlaps physically the DATA and BIT areas.
IDATALEN        EQU     100H
;
;  XDATASTART: XDATA memory start address <0x0-0xFFFF>
;       The absolute start address of XDATA memory
XDATASTART      EQU     0
;
;  XDATALEN: XDATA memory size <0x0-0xFFFF>
;       The length of XDATA memory in bytes.
XDATALEN        EQU     0
;
;  PDATASTART: PDATA memory start address <0x0-0xFFFF>
;       The absolute start address of PDATA memory
PDATASTART      EQU     0H
;
;  PDATALEN: PDATA memory size <0x0-0xFF>
;       The length of PDATA memory in bytes.
PDATALEN        EQU     0H
;
;;------------------------------------------------------------------------------
;
;  Reentrant Stack Initialization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialized it:
;
; Stack Space for reentrant functions in the SMALL model.
;   IBPSTACK: Enable SMALL model reentrant stack
;       Stack space for reentrant functions in the SMALL model.
IBPSTACK        EQU     0       ; set to 1 if small reentrant is used.
;  IBPSTACKTOP: End address of SMALL model stack <0x0-0xFF>
;       Set the top of the stack to the highest location.
IBPSTACKTOP     EQU     0xFF +1         ; default 0FFH+1
;
;
; Stack Space for reentrant functions in the LARGE model.
;  XBPSTACK: Enable LARGE model reentrant stack
;       Stack space for reentrant functions in the LARGE model.
XBPSTACK        EQU 0           ; set to 1 if large reentrant is used.
;  XBPSTACKTOP: End address of LARGE model stack <0x0-0xFFFF>
;       Set the top of the stack to the highest location.
XBPSTACKTOP     EQU     0xFFFF +1       ; default 0FFFFH+1
;
;
; Stack Space for reentrant functions in the COMPACT model.
```

```
;  PBPSTACK: Enable COMPACT model reentrant stack
;       Stack space for reentrant functions in the COMPACT model.
PBPSTACK       EQU     0       ; set to 1 if compact reentrant is used.
;
;  PBPSTACKTOP: End address of COMPACT model stack <0x0-0xFFFF>
;       Set the top of the stack to the highest location.
PBPSTACKTOP    EQU     0xFF +1         ; default 0FFH+1
;
;;------------------------------------------------------------------------------
;
;  Memory Page for Using the Compact Model with 64 KByte xdata RAM
;  Compact Model Page Definition
;
;  Define the XDATA page used for PDATA variables.
;  PPAGE must conform with the PPAGE set in the linker invocation.
;
; Enable pdata memory page initalization
PPAGEENABLE    EQU     0       ; set to 1 if pdata object are used.
;
; PPAGE number <0x0-0xFF>
; uppermost 256-byte address of the page used for PDATA variables.
PPAGE   EQU     0
;
```

```
; SFR address which supplies uppermost address byte <0x0-0xFF>
; most 8051 variants use P2 as uppermost address byte
PPAGE_SFR       DATA    0A0H
;
;;-------------------------------------------------------------------------------

; Standard SFR Symbols
ACC     DATA    0E0H
B       DATA    0F0H
SP      DATA    81H
DPL     DATA    82H
DPH     DATA    83H

                NAME    ?C_STARTUP
?C_C51STARTUP   SEGMENT         CODE
?STACK          SEGMENT         IDATA
                RSEG    ?STACK
MAIN_STACK:     DS      1
                EXTRN CODE      (?C_START)
                PUBLIC          ?C_STARTUP
                                PUBLIC MAIN_STACK
                CSEG    AT      0
?C_STARTUP:     LJMP    STARTUP1
                RSEG    ?C_C51STARTUP

STARTUP1:

IF IDATALEN <> 0
                MOV     R0,#IDATALEN - 1
                CLR     A
IDATALOOP:      MOV     @R0,A
                DJNZ    R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
                MOV     DPTR,#XDATASTART
                MOV     R7,#LOW (XDATALEN)
  IF (LOW (XDATALEN)) <> 0
                MOV     R6,#(HIGH (XDATALEN)) +1
  ELSE
                MOV     R6,#HIGH (XDATALEN)
  ENDIF
                CLR     A
XDATALOOP:      MOVX    @DPTR,A
                INC     DPTR
                DJNZ    R7,XDATALOOP
                DJNZ    R6,XDATALOOP
```

```
ENDIF

IF PPAGEENABLE <> 0
               MOV     PPAGE_SFR,#PPAGE
ENDIF

IF PDATALEN <> 0
               MOV     R0,#LOW (PDATASTART)
               MOV     R7,#LOW (PDATALEN)
               CLR     A
PDATALOOP:     MOVX    @R0,A
               INC     R0
               DJNZ    R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
               MOV     ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
               MOV     ?C_XBP,#HIGH XBPSTACKTOP
               MOV     ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
               MOV     ?C_PBP,#LOW PBPSTACKTOP
ENDIF

               MOV     SP,#?STACK-1

; This code is required if you use L51_BANK.A51 with Banking Mode 4
; Code Banking
; Select Bank 0 for L51_BANK.A51 Mode 4
#if 0
;       <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with
Banking Mode 4.
EXTRN CODE (?B_SWITCH0)
               CALL    ?B_SWITCH0     ; init bank mechanism to code bank 0
#endif
;              LJMP    ?C_START

               END
```

ParrOS.a51

```
;  ParrOS.a51
;  STORES ALL TASK REGISTERS
;
```

```
; ================================================================
; EACH TASK CAN BE MADE TO USE ANY NUMBER OF TIME SLOTS (1 TO 255)
; SO THAT NOT ALL TASKS RUN FIOR THE SAME AMOUNT OF TIME.
; NOMINALLY THEY RUN FOR JUST ONE TIME SLOT
; ================================================================
;

; INCLUDES RTOSGOSEC FOR 1 SECOND TICKS
;
; LATEST - HANDLES 20 TASKS OR MORE, DEPENDING ON
; EXTERNAL MEMORY AND INTERNAL STACK SPACE
; CAN BE USED WITH ASSEMBLY LANGUAGE MAIN PROGRAM
;
; Written by Paul P. Debono – NOVEMBER 2002
; University of Malta
; Department of Communications and Computer Engineering
; MSIDA MSD 2080; MALTA.
; Adapted and modified from the RTKS RTOS FOR THE 8032 BOARD

; Accomodates 20 OR MORE tasks, (take care of the stack size!)
; STACK MOVING VERSION - MOVES WORKING STACK IN AND OUT OF
; EXTERNAL MEMORY
; SLOWS DOWN RTOS, BUT DOES NOT RESTRICT TASK CALLS
;
; Uses timer 2, in 16-bit auto-reload mode as the time scheduler (time-ticker)
; All tasks run in bank 0, RTOS kernel runs in bank 1
; All tasks must be written as an endless loop.
;
; IDLE TASK (ENDLESS MAIN PROGRAM HAS A TASK NUMBER = NOOFTASKS)
;
; COMMANDS AVAILABLE FOR THE C APPLICATION PROGRAM ARE:
; (valid parameter values are shown in parenthesis)
;
; INIT_RTOS(TSLOT)            Initialise variables with default Tslot (for Main) (1-255)
; CREATE(TSK#,TSLOT,TSKADDR)  Create a new task.
;                             TSK# passed in R7 BANK 0
;                             TSLOT passed in R5 BANK 0
;                             TSKADDR in R1 (low byte) and R2 (high byte) BANK 0
; RTOSGOMSEC(TICKTIME)        Start RTOS going, interrupt every TICKTIME msecs (1-255).
;
; THIS IS STILL A SMALL TEST VERSION RTOS. IT IS JUST USED FOR
; SHOWING WHAT IS NEEDED TO MAKE A SIMPLE RTOS.
; IT MIGHT STILL NEED SOME MORE FINE TUNING.
; IT HAS NOT BEEN NOT THOROUGHLY TESTED !!!!
; WORKS FINE SO FAR.
; NO RESPONSABILITY IS TAKEN.
```

```
$NOMOD51
#include "reg52.h"     ; check your own correct path
USING 1
;  ASSEMBLER MACROS
SetBank MACRO BankNumber
IF BankNumber = 0
        CLR RS0
        CLR RS1
ENDIF
IF BankNumber = 1
  SETB RS0
  CLR RS1
ENDIF
ENDM

Ext2Int MACRO          ; MOVES R0 DATA FROM EXT DPTR POINTER TO INTERNAL R1 POINTER
  MOV R1,#MAIN_STACK
  MOV R0,#STACKSIZE
NEXT11:
  MOVX A,@DPTR
  MOV @R1,A
  INC DPTR
  INC R1
  DJNZ R0,NEXT11
ENDM

Int2Ext MACRO          ; MOVES R0 DATA FROM INTERNAL R1 POINTER TO EXT DPTR POINTER
                       ; USES R0, R1, ACC AND DPTR
  MOV R1,#MAIN_STACK
  MOV R0,#STACKSIZE
NEXT12:
  MOV A,@R1
  MOVX @DPTR,A
  INC DPTR
  INC R1
  DJNZ R0,NEXT12
ENDM

Push_Bank0_Reg MACRO
  PUSH ACC
  PUSH B
  PUSH PSW
  PUSH DPL
  PUSH DPH
  PUSH 00
  PUSH 01
```

```
  PUSH 02
  PUSH 03
  PUSH 04
  PUSH 05
  PUSH 06
  PUSH 07
ENDM

Pop_Bank0_Reg MACRO
  POP 07
  POP 06
  POP 05
  POP 04
  POP 03
  POP 02
  POP 01
  POP 00
  POP DPH
  POP DPL
  POP PSW
  POP B
  POP ACC
ENDM
```

```
;
; NOTE: Functions which receive parameters when
;       called from within C must have their name
;       start with an underscore in the A51 source file.
;

PUBLIC _RTOSGOMSEC, _RTOSGOSEC, _RTOSGOMIN
PUBLIC _CREATE, _INIT_RTOS

CLOCK          EQU 460        ; COUNT FOR HALF A MILLISECOND
; timer clock (11059/12 = 922) counts for 1 msec assuming 11.0592 MHz crystal
; hence 921.6/2 = 460 for half a milli second
BASIC_TICK EQU 65535 - CLOCK + 1
ONEMSEC        EQU 2          ; 2 HALF MSECS EQUAL 1 MSEC
ONESEC         EQU 2000       ; 2000 1/2 MSEC TICKS = 1 SECOND
HALFMIN        EQU 60000      ; 60000 1/2 MSEC TICKS = 1/2 MINUTE

XTRAMTOP       EQU 0FFFFH     ; FLT32 EXTERNAL RAM TOP
RAMTOP         EQU 0FFH       ; MAXIMUM VALUE FOR 8032 WOULD BE 0FFH
NOOFTSKS       EQU 16         ; CAN HAVE MORE TASKS (numbered 0 to N-1)
;
; ****************************************************************************
;                      I M P O R T A N T
; THIS IS REQUIRED SO THAT THE LOCATION OF THE STACK IS KNOWN
; THIS IS TAKEN FROM THE VALUE WORKED OUR IN PARROS_STARTUP.A51
;
EXTRN IDATA (MAIN_STACK)
;
; ****************************************************************************
;                             ; LIMITED ONLY BY STACK/MEMORY SPACE
STACKSIZE EQU 30H             ; 15H MINIMUM
NOOFPUSHES EQU 13             ; NUMBER OF PUSHES AT BEGINNING OF RTOS_INT ROUTINE
                              ; WITH LESS TASKS, YOU CAN INCREASE STACKSIZE
                                      ; SIZE OF STACK IS CRITICAL AND SYSTEM CAN CRASH
                                      ; IF YOU USE A LARGE OR EVEN A SMALLER VALUE. TRY IT OUT
NOT_TIMING EQU 0FFH
IDLE_TASK EQU NOOFTSKS        ; main endless loop in C application given
                                      ; a task number equal to NOOFTSKS

MYBITS SEGMENT BIT
RSEG MYBITS
  MSECFLAG: DBIT 1            ; MARKER TO INDICATE TICKS EVERY X MILLISECONDS
  SECFLAG: DBIT 1             ; MARKER TO INDICATE TICKS EVERY X SECONDS
  MINFLAG: DBIT 1                     ; MARKER TO INDICATE TICKS EVERY X MINUTES

VAR1 SEGMENT DATA
RSEG VAR1                     ; VARIABLE DATA AREA VAR1,
```

```
                                ; range 0x10-0xFF, since we are using Banks 0,1
;DSEG AT 10H
 READYQTOP:   DS 1           ; ADDRESS OF LAST READY TASK
 RUNNING:     DS 1           ; NUMBER OF CURRENT TASK
 TICKCOUNT:   DS 1                  ; USED FOR RTOSGO.....
 DELAYLO:     DS 1                  ; USED FOR RTOSGO.....
 DELAYHI:     DS 1                  ; USED FOR RTOSGO.....
 GOPARAM:     DS 1                  ; USED FOR RTOSGO.....
 TMPSTORE0:   DS 1           ; USED IN FETCHSTACK

DSEG AT 22H
 READYQ:      DS (NOOFTSKS + 1)     ; QUEUE STACK FOR TASKS READY TO RUN
 SPTS:        DS (NOOFTSKS + 1)     ; SP FOR EACH TASK AND 1 FOR THE IDLE TASK
 T_SLOT:      DS (NOOFTSKS + 1)     ; TIME SLOTS USAGE PER TASK AND MAINS
 T_SLOT_RELOAD: DS (NOOFTSKS + 1)   ; RELOAD VALUE FOR TIME SLOTS ABOVE
; MAIN_STACK AREA STARTS HERE, NEXT LOCATION AFTER TSKFLAGS.

;SPARE_STACK SEGMENT XDATA          ; VARIABLE EXTERNAL DATA
;RSEG SPARE_STACK
XSEG AT 1 + XTRAMTOP - (NOOFTSKS + 1) * STACKSIZE
 EXT_STK_AREA: DS (NOOFTSKS + 1) * STACKSIZE   ; THIS IS THE ACTUAL SIZE OF STACK AREA

;CSEG AT 8028H                      ; INTERRUPT VECTOR ADDRESS FOR TIMER 2 ON FLIGHT
32 BOARD
CSEG AT 0028H                       ; INTERRUPT VECTOR ADDRESS FOR TIMER 2 ON GENERIC
CONTROLLER
 CLR EA                             ;
 CLR TF2                            ; Clear Timer 2 interrupt flag (not done
automatically)
 LJMP RTOS_TIMER_INT

MyRTOS_CODE SEGMENT CODE            ; STARTS AT 8100H FOR THE FLIGHT32 BOARD
RSEG MyRTOS_CODE

; START OF RTOS SYSTEM
; PREFIX NAME FOR FUNC WITH REG-PASSED PARAMS MUST START WITH AN UNDERSCORE _

_INIT_RTOS:                         ; SYS CALL TO SET UP VARIABLES
                                    ; R7 HOLDS THE DEFAULT TSLOT (FOR MAIN)
; IN THE C ENVIRONMENT, THE KEIL SOFTWARE CLEARS THE INTERNAL RAM FROM 0 TO 7FH
; WHEN THE STARTUP SEQUENCE IS CALLED.
; EVEN THOUGH THE 8032 WITH 0-FFH INTERNAL RAM WAS CHOSEN IN THE TARGET OPTION.
; HENCE CERTAIN VARIABLES STORED FROM 80H TO FFH (SUCH AS TSKFLAGS) MUST BE
; INITIALISED TO ZERO IN THIS INITALISATION ROUTINE.
;
; IN ASM OR A51 (NOT IN C), ALL THE INTERNAL RAM (0-FFH) IS
; CLEARED BY MEANS OF THE CLR_8051_RAM MACRO.
;
```

```
  MOV DPTR,#EXT_STK_AREA              ; NEXT CLEAR ALL EXTERNAL RAM STACKS
  MOV R0,#(NOOFTSKS + 1)
  CLR A
NEXT_STACK:
  MOV R1,#STACKSIZE
CLR_STACK:
  MOVX @DPTR,A
  INC DPTR
  DJNZ R1,CLR_STACK
  DJNZ R0,NEXT_STACK
  MOV R5,07                           ; STORE DEFAULT TSLOT IN R5

  MOV IE,#20H                         ; ENSURE EA = 0 AND ET2 = 1
  MOV RUNNING,#IDLE_TASK              ; IDLE TASK RUNNING (Main program endless loop)
  MOV R7,#(NOOFTSKS + 1)               ; FILL ONE ADDITIONAL LOCATION, FOR MAIN IDLE
TASK
  MOV R1,#READYQ
LOAD_VARS:
  MOV @R1,#IDLE_TASK                  ; IDLE TASK IN ALL OF READYQ (Main program end-
less loop)
  INC R1
  DJNZ R7,LOAD_VARS                   ; SET UP ALL TASKS
  MOV READYQTOP,#READYQ
```

```
; INITIALISE ALL STACK POINTERS
 MOV R7,#NOOFTSKS                    ; COUNTER
 MOV R0,#SPTS                        ; MAIN IDLE TASK TAKEN CARE OF BY 1ST INTERRUPT
 MOV A,#(MAIN_STACK - 1)
 ADD A,#(NOOFPUSHES + 2)
SET_UP:
 MOV @R0,A                           ; ALL SPs POINT TO MAIN_STACK + PUSHES, IN
PREPARATION
 INC R0                              ; FOR THE EVENTUAL RETI INSTRUCTION
 DJNZ R7,SET_UP                      ; USED TO CHANGE TASKS AFTER AN RTOS INTERRUPT.

; INITIALISE TIME SLOTS, INITIALLY ALL SET TO THE GIVEN DEFAULT VALUE
 MOV R7,#(NOOFTSKS +1)
 MOV R0,#T_SLOT
 MOV R1,#T_SLOT_RELOAD
LOAD_SLOTS:
 MOV @R0,05
 MOV @R1,05
 INC R0
 INC R1
 DJNZ R7,LOAD_SLOTS
 RET

_CREATE:
                                     ; SYS CALL ENTRY TO CREATE A TASK
                                     ; TASK NUMBER (0 to 7) PASSED IN BANK0 R7
                                     ; TIME SLOT (1 - 255) PASSED IN BANK0 R5
                                     ; TASK START ADDR PASSED IN BANK0 R1,R2,R3
                                     ; LSB in R1, MSB in R2, R3 contains type
 INC READYQTOP
 MOV R0, READYQTOP
 MOV @R0,07                          ; PLACE TASK IN READYQ
 MOV A,#T_SLOT
 ADD A,R7
 MOV R0,A
 MOV @R0, 05                         ; PUT GIVEN TIME SLOT (IN R5) INTO MEM LOCATION
 MOV A,#T_SLOT_RELOAD
 ADD A,R7
 MOV R0,A
 MOV @R0, 05                         ; PUT GIVEN TIME SLOT (IN R5) INTO RELOAD
                                     ; MEM LOCATION (T_SLOT_RELOAD)
 MOV A,R7
 CALL FetchStack
 MOV A,R1
 MOVX @DPTR,A                        ; copy low byte R1 into LOW STACK AREA
 INC DPTR
```

```
  MOV A,R2
  MOVX @DPTR,A                    ; NOW SAVE THE HIGH ORDER BYTE (R2)
  RET

_RTOSGOMSEC:                      ; SYS CALL TO START RTOS FOR R7 MILLISECOND TICKS
  SETB MSECFLAG                   ; SET MARKER
  CLR SECFLAG
  CLR MINFLAG
  MOV DELAYLO,#LOW(ONEMSEC)
  MOV DELAYHI,#HIGH(ONEMSEC)
  SJMP LOAD_REGS

_RTOSGOSEC:                       ; SYS CALL TO START RTOS FOR R7 SECOND TICKS
  SETB SECFLAG                    ; SET MARKER
  CLR MINFLAG
  CLR MSECFLAG
  MOV DELAYLO,#LOW(ONESEC)         ; EQUAL ONE SECOND
  MOV DELAYHI,#HIGH(ONESEC)
  SJMP LOAD_REGS

_RTOSGOMIN:
  SETB MINFLAG
  CLR MSECFLAG
  CLR SECFLAG
  MOV DELAYLO,#LOW(HALFMIN)       ; 60000 HALF MILLISECONDS EQUAL HALF MINUTE
  MOV DELAYHI,#HIGH(HALFMIN)

LOAD_REGS:
  MOV RCAP2H,#HIGH(BASIC_TICK)    ; LOAD RCAPS WITH 1 MILLISECOND COUNT
  MOV RCAP2L,#LOW(BASIC_TICK)            ; SAVE THEM IN THE AUTO RE-LOAD REGISTERS
                                  ; OF TIMER 2 (for Flight 32)
  MOV GOPARAM,07                  ; LOAD TICKS PARAMETER, PASSED IN R7 BANK 0
  MOV TICKCOUNT,07
  MOV T2CON,#04H                  ; START TIMER 2 IN 16-BIT AUTO RELOAD MODE.
  SETB EA                         ; ENABLE GLOBAL INTERRUPT SIGNAL
  SETB TF2                        ; SIMULATE TIMER 2 INTERRUPT
  RET                             ; EFFECTIVELY STARTING THE RTOS.

EXIT1: LJMP EXIT                  ; STEPPING STONE

RTOS_TIMER_INT:                   ; INTERRUPT ENTRY ONLY FROM TIMER2 OVERFLOW
INTERRUPT

                                  ; USES ACC,PSW, (R0,R1 AND R2 FROM BANK 1)
  Push_Bank0_Reg
  SetBank 1                       ; SET TO REGISTERBANK 1
  CLR C
  MOV A, DELAYLO
```

```
 SUBB A,#1
 MOV DELAYLO,A
 MOV A,DELAYHI
 SUBB A,#0
 MOV DELAYHI,A
 ORL A,DELAYLO                   ; CHECK IF DELAY (1 MSEC, 1 SEC OR 1/2 MIN) HAS
PASSED
 JNZ EXIT                               ; IF NOT, EXIT
 MOV DELAYLO,#LOW(ONEMSEC)       ; DELAY OF 1 MSECS
 MOV DELAYHI,#HIGH(ONEMSEC)
 JB MSECFLAG,CHK_GO_PARAM
 MOV DELAYLO,#LOW(ONESEC)
 MOV DELAYHI,#HIGH(ONESEC)
 JB SECFLAG,CHK_GO_PARAM
 MOV DELAYLO,#LOW(HALFMIN)
 MOV DELAYHI,#HIGH(HALFMIN)
 CPL MINFLAG
 JNB MINFLAG, EXIT1              ; WAIT FOR ONE MINUTE (TWICE HALF MIN)
CHK_GO_PARAM:
 DJNZ TICKCOUNT, EXIT1            ; CHECK IF REQUIRED TIME SLOTS HAVE PASSED
 MOV TICKCOUNT,GOPARAM
                                 ; QROTATE
                                 ; SAVE PRESENT RUNNING TASK STACK PTR
                                 ; ROTATE READYQ BY ONE
                                 ; GET NEW RUNNING TASK FROM READYQ
 MOV A,#T_SLOT                   ; FIRST CHECK IF REQUIRED TIME SLOT HAS PASSED
 ADD A,RUNNING
 MOV R0,A                        ; R0 POINTS TO T_SLOT OF RUNNING TASK
 MOV A,@R0
 DEC A
 MOV @R0,A                       ; SAVE DECREMENTED TIME SLOT
 JNZ EXIT1                       ; TIME SLOT NOT FINISHED, HENCE EXIT WITHOUT
                                 ; CHANGING TASKS
  MOV A,#T_SLOT_RELOAD           ; TIME SLOT PASSED, THEREFORE RELOAD WITH
 ADD A,RUNNING                   ; ORIGINAL VALUE AND CHANGE TASKS
 MOV R1,A                        ; R1 POINTS TO T_SLOT_RELOAD OF RUNNING TASK
 MOV A,@R1
 MOV @R0,A                       ; RESET ORIGINAL TIME SLOT VALUE

 MOV A, #SPTS                    ; save SP
 ADD A, RUNNING
 MOV R0,A
 MOV @R0, SP                     ; store present stack pointer of task.
 MOV A,RUNNING
 MOV R5, A                       ; SAVE CURRENT TASK IN R5 BANK 1 (ADDRESS 0DH)
 CALL FetchStack
```

```
  Int2Ext                             ; SAVE STACK IN EXTERNAL, READY FOR SWAP
  MOV R1,#(READYQ + 1)                ; Now SHIFT Q DOWN 1
  MOV R0, READYQTOP
  DEC R0                              ; R0 NOW POINTS TO ONE BYTE BELOW TOP OF QUEUE
SHIFT_DOWN:
  MOV A,@R1
  DEC R1
  MOV @R1,A
  MOV A,R1
  INC R1
  INC R1
  CJNE A,08,SHIFT_DOWN                ; THEY ALL MOVED DOWN SO
  INC R0                              ; R0 NOW POINTS AGAIN TO READYQTOP
  MOV @R0, 0DH                        ; PLACE CURRENT TASK ON TOP OF QUEUE
RUN_NEW_TASK: ; run new task
  MOV A,READYQ
  MOV RUNNING,A                       ; SET NEW TASK AS RUNNING
  CALL FetchStack
  Ext2Int                             ; GET NEW STACK IMAGE
  MOV A,#SPTS
  ADD A,RUNNING
  MOV R0,A
  MOV SP,@R0                          ; SET SP TO NEW TASK STACK AREA
```

```
EXIT:
  Pop_Bank0_Reg
  SetBank 0
  SETB EA
  RETI
; SUB ROUTINES

FetchStack:
; ENTRY A = TASK NUMBER, USES ACC, DPTR, AND R0
; EXIT DPTR POINT TO START OF STACK AREA FOR TASK
  PUSH 00
  PUSH 08
  MOV TMPSTORE0,A
  MOV DPTR,#EXT_STK_AREA
  MOV R0,#0
LOOP1:
  MOV A,R0
  CJNE A,TMPSTORE0,CONT1
  POP 08
  POP 00
  RET
CONT1:
  MOV A,#STACKSIZE
  ADD A,DPL
  MOV DPL,A
  MOV A,DPH
  ADDC A,#0
  MOV DPH,A
  INC R0
  SJMP LOOP1
END
```

Appendix B PaulOS A51 version

The PaulOS RTOS

This is the A51 assembly language version of the PaulOS (PAUL's Operating System) RTOS. It has been superseded by its C language version but we have included it here for the benefit of those who are keen to use the assembly language even in 'large' projects. Most of the explanations have already been included in Chapter 9 but are being retained here so as to make it a self-contained appendix. The idea behind the PaulOS RTOS is that any task (part of program) can be in any ONE of three states:

RUNNING

It can be RUNNING, (obviously in the single 8051 environment, there can only be one task which is running.)

WAITING

It can be in the WAITING or SLEEPING queue. Here a task could be waiting for any one of the following:

- a specified amount of time, selected by the user with WAITT command.
- a specified amount of time, selected by the user with PERIODIC command.
- a specified interrupt to occur within a specified time, selected by the user with the WAITI command.
- a signal from some other task within a specified timeout.
- a signal from some other task indefinitely.
- finally, a task could be waiting here for ever, effectively behaving as if the task did not exist. This is specified by the KILL command.

READY

It can also be in the READY QUEUE, waiting for its turn to execute.

This can be visualised in Figure 7-1 which shows how the task can move from one state to the other.

The RTOS itself always resides in the background, and comes into play:

- At every RTOS TIMER interrupt (usually Timer 2 or Timer 0, every one millisecond).
- At any other interrupt from other timers or external inputs.
- Whenever an RTOS system command is issued by the main program or tasks.

The RTOS which is effectively supervising all the other tasks, then has to make a decision whether it has to change tasks. There could be various reasons for changing tasks, as explained further on, but in order to do this task swap smoothly, the RTOS has to save all the environment of the presently running task and substitute it with the environment of the next task which is about to run. This is accomplished by saving all the BANK 0 registers, the ACC, B, PSW, and DPTR registers. The STACK too has to be saved since the task might have pushed some data on the stack (apart from the address in the task program, where it has to return to after the interrupt).

System Commands

Here is a detailed explanation of all the PaulOS RTOS system commands. They are listed in the sequence in which they appear in the PaulOS.A51 source program. Note that certain system commands initiate a task change whilst others do not.

The following calls listed in Table B-2 do not receive parameters, hence are not declared with an underscore prefix in the a51 file.

void SET_IDLE_MODE(void)	- Normally used in Main idle task
void SET_POWER_DOWN(void)	- Normally used in Main idle task
void DEFER(void)	- This commands causes a task change
void KILL(void)	- This commands causes a task change
uchar SCHEK(void)	
uchar RUNNING_TASK_ID(void)	
void WAITV(void)	- This commands causes a task change

Table B-2 System Calls without any parameters

The following calls listed in Table B-3 do require parameters, hence are declared with an underscore prefix in the a51 file.

void INIT_RTOS(uchar IEMASK)	- Normally used in Main idle task
void RTOSGOMSEC(uchar msecs,uchar prior)	- Normally used in Main idle task
void SIGNAL(uchar task)	
void WAITI(uchar intnum)	- This commands causes a task change
void WAITT(uint ticks)	- This commands causes a task change
void WAITS(uint ticks)	- This commands causes a task change if signal is not yet present
void CREATE(uchar task,uint *taskadd)	- Normally used in Main idle task
void PERIODIC(uint ticks)	
void RESUME(uchar task)	- This commands causes a task change

Table B-3 System calls needing some parameters

INIT_RTOS(IEMASK)

This system command must be the FIRST command to be issued in the main program in order to initialise the RTOS variables. It is called from the main program and takes the interrupt enable mask (IEMASK) as a parameter. An example of the syntax used for this command is:

```
INIT_RTOS(0x30);
```

which would imply that some task is intended to use the Timer 2 interrupt (IEMASK=20H) for the RTOS as well as the Serial Interrupt (IEMASK=10H). (See Table B-4). The default mask is 20H which enables just the Timer 2 interrupt. This 20H is always added (or ORed) by the RTOS automatically to any other mask. Other masks which are valid are:

Interrupt		IE MASK		Notes
No:	Name	Binary	Hex	
0	External Int 0	00000001	01	
1	Timer Counter 0	00000010	02	Default RTOS for 8051
2	External Int 1	00000100	04	
3	Timer Counter 1	00001000	08	
4	Serial Port	00010000	10	
5	Timer 2 (8032 only)	00100000	20	Default RTOS for 8032

Table B-4 IEMASK parameter

This system command performs the following:

- Clears the external memory area which is going to be used to store the stack of each task.
- Sets up the IE register (location A8H in the SFR area)}.
- Selects edge triggering on the external interrupts. (can be amended if different triggering required).
- Loads the Ready Queue with the main idle task number, so that initially, only the main task will execute.
- Initialises all task as being not waiting for a timeout.
- Sets up the SP of each task to point the correct location in the stack area of the particular task. The stack pointer, initially, is made to point to an offset of 14 above the base of the stack [(MAIN_STACK - 1) + NOOFPUSHES + 2] since NOOFPUSHES in this case is 13. This is done so as to ensure that when the first RET instruction is executed after transferring the stack from external RAM on to the 8032 RAM, the SP would be pointing correctly to the address of the task to be started. This is seen in the QSHFT routine, where before the last RET instruction, there is the Pop_Bank0_Reg macro which effectively pops 13 registers. The RET instruction would then read the correct address to jump to from the next 2 locations.

CREATE(Task No:, Task Name)

This system command is used in the main program for each task to be created. It takes two parameters, namely the task number (1st task is numbered as 0), and the task address, which in the C environment, would simply be the name of the procedure. An example of the syntax used for this command is:

CREATE(0,MotorOn);

This would create a task, number 0. This task would in fact be the MotorOn procedure.

This system command performs the following:

- Places the task number in the next available location in Ready Queue, meaning that this task is ready to execute. The location pointer in Ready Queue is referred to as READYQTOP in the program, and is incremented every time this command is issued.
- Loads the address of the start of the task in the bottom of the stack area in external ram allocated to this task. The SP for this task would have been already saved, by the INIT_RTOS command, pointing to an offset 13 bytes above this.

RTOSGOMSEC(Msec, Priority)

This system command is used only ONCE in the main program, when the RTOS would be required to start supervising the processes. It takes two parameters, namely:

The number of milliseconds, which would be the base reference for other time dependent commands, such as PERIODIC, WAITT and WAITS.

The Priority (0 or 1), which if set to 1, implies that tasks placed in the Ready Queue, ready to execute, would be sorted in descending order before the RTOS selects the next task to run. A task number of 0 is assigned to the HIGHEST priority task, and would obviously be given preference during the sorting.

An example of the syntax used for this command is:

RTOSGOMSEC(10,1)

This would start the RTOS ticking, at a reference time signal of 10 milliseconds. This 10 milliseconds would then become the basic reference unit for other system commands which use any timeout parameter.

The RTOS would also be required to execute task sorting prior to any task change. It should be pointed out here, that the RTOS timer would still be generating interrupts every half a millisecond (if the HALFMSEC variable is set to 1 in the file) , so as to respond to external interrupts relatively quickly.

This system command performs the following:

- Loads the variable DELAY (LO and HI bytes), with the number of BASIC_TICKS required to obtain a one millisecond interval. Since BASIC_TICKS correspond to a half second interval in Timer 2, then to get a one millisecond interval, DELAY is simply loaded with 2.
- Set the PRIORITY bit according to the priority parameter supplied.
- Load RCAP2H and RCAP2L, the timer 2 registers, with the required count in order to obtain half a millisecond interval between timer 2 overflow interrupts. The value used depends on the crystal frequency used on the board. The clock registers count up at one twelfth the clock frequency, and using a clock frequency of 11.0592 MHz, each count would involve a time delay of 12/11.0592 µsec. (1.085 µsec).
- Therefore to get a delay of half a millisecond (500 µsecs), 500/1.085 or 460.8 counts would be needed. Since there are a lot of overheads in the Pushes and Pops involved during every interrupt, a count of 450 was used. Moreover, since the timers generate an interrupt when there is an overflow in the registers, then the registers are actually loaded with 65086 or (65536 – 450).
- Store the reference time signal parameter in GOPARAM and TICKCOUNT.
- Start timer 2 in 16-bit auto-reload mode.
- Enable interrupts.
- Set TF2, which is the timer 2 overflow interrupt flag, thus causing the 1st interrupt.

RUNNING_TASK_ID( )

This system command is used by a task to get the number of the task itself. It returns a value (in R7 bank 0). The same task continues to run after executing this system command.

An example of the syntax used for this command is:

X = RUNNING_TASK_ID(); /* where X would be an unsigned integer */

SCHEK( )

This system command is used by a task to test whether there was any signal sent to it by some other task. It returns a value (in R7 bank 0):

1 - Signal is not present
0 - Signal is present

If the signal was present, it is also cleared before returning to the calling task. The same task continues to run, irrespective of the returned value.

An example of the syntax used for this command is:

```
X = SCHEK(); /* where X would be an unsigned integer */
```

or you may use it in the following example to test the presence of the signal bit:

```
if (SCHEK() == 0)

 {
 /* do these instructions if a signal was present */
 }
```

SIGNAL(Task No:)

This system command is used by a task to send a signal to another task. If the other task was already waiting for a signal, then the other task is placed in the Ready Queue and its waiting for signal flag is cleared. The task issuing the SIGNAL command continues to run, irrespective of whether the called task was waiting or not waiting for the signal. If you need to halt the task after the SIGNAL command to give way to other tasks, you must use the DEFER() system command after the SIGNAL command.

This system command performs the following:

- It first checks whether the called task was already waiting for a signal.
- If the called task was not waiting, it set its waiting for signal (SIGW) flag and exits to continue the same task.
- If it was already waiting, it places the called task in the Ready Queue and it clears both the waiting for signal (SIGW) and the signal present (SIGS) flags.
- It also sets a flag (TINQFLAG) to indicate that a new task has been placed in the Ready Queue. This flag is used by the RTOS_TIMER_INT routine (every half a millisecond) in order to be able to decide whether there has to be a task change. It then exits the routine to continue the same task.

An example of the syntax used for this command is:

SIGNAL(1); // send a signal to task number 1

The following commands perform a change of task:

WAITI(Interrupt No:)

This system command is called by a task to 'sleep' and wait for an interrupt to occur. Another task, next in line in Ready Queue would then take over. If the interrupt never occurs, then the task will effectively sleep for ever.

If required, this command can be modified to allow another timeout parameter to be passed, so that if the interrupt does not arrive within the specified timeout, the task would resume. A timeout of 0 would leave the task still waiting the interrupt forever. This would be similar to the WAITS command explained further down.

This system command performs the following:

- It sets the bit which correspond to the interrupt number passed on as a parameter.
- It then calls the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

WAITI(0); // wait for an interrupt from external int 0

The task would then go into the sleep or waiting mode and a new task would take over.

WAITS(Timeout)

This system command is called by a task to sleep and wait for a signal to arrive from some other task. If the signal is already present (previously set by some other task), then the signal is simply cleared and the task continues on. If the signal does not arrive within the specified timeout period, the task resumes just the same. However, a timeout number of 0 would imply that the task has to keep on waiting for a signal indefinitely. If the signal does not arrive, then the task never resumes to work and effectively kills the task.

This system command performs the following:

- It first checks whether the signal is already present.
- If it is it clears the signal flag, exits and continues running
- If signal is not present, then:
- It sets its own waiting for signal (SIGW) flag.
- It also sets the waiting for timeout variable according to the supplied parameter.
- It then jumps to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

WAITS(50);
// wait for a signal for 50 units, the value of the unit depends on // the RTOSGOMSEC parameter used.

If for example, the command RTOSGOMSEC(10,1) was used, the reference unit would be 10 milliseconds, and WAITS(50) would then imply waiting for a signal to arrive within 500 milliseconds.

or you can use:

WAITS(0); // this would wait for a signal for ever

In both examples, the task would then go into the sleep or waiting mode and a new task would take over.

WAITT(Timeout)

This system command is called by a task to sleep and wait for a specified timeout period. The timeout period is in units whose value depends on the RTOSGOMSEC parameter used. Valid values for the timeout period are in the range 1 to 655635. A value of 0 is reserved for the KILL command, meaning permanent sleep, and therefore is not allowed for this command. The WAITT system command therefore performs the required check on the parameter before accepting the value. A value of 0 is changed to a 1.

This system command performs the following:

- If the parameter is 0, then set it to 1, to avoid permanent sleep.
- Saves the correct parameter in its correct place in the TTS table.
- Jumps to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

WAITT(60);
// wait for a signal for 60 units, the value of the unit depends on // the RTOSGOMSEC parameter used.

If for example, the command RTOSGOMSEC(10,1) was used, the reference unit would be 60 milliseconds, and WAITT(60) would then imply waiting or sleeping for 600 milliseconds.

If on the other hand, the command RTOSGOMSEC(250,1) was used, the reference unit would be a quarter of a second, and WAITT(240) would then imply waiting or sleeping for 60 seconds or 1 minute.

In both examples, the task would then go into the sleep or waiting mode and a new task would take over.

KILL( )

This system command is used by a task in order to stop or terminate the task. As explained earlier in WAITT, this is simply the command WAITT with an allowed timeout of 0. The task is then placed permanently waiting and never resumes execution.

This system command performs the following:

- It first clears any waiting for signal or interrupt flags, so that that task would definitely never restart.
- It then sets its timeout period in the TTS table to 0, which is the magic number the RTOS uses to define any non-timing task.
- It then sets the INTVLRLD and INTVLCNT to 0, again implying not a periodic task.
- Jumps to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

KILL();
/* the task simply stops to execute and a new task would take over.*/

RESUME (Task Number)

This system command is used in a task to resume another task which had already KILLed itself. The parameter passed is the task number of the task which has to be restarted. After executing this command, the calling task itself is DEFERred to give up its CPU time to any other task (presumably the resurrected task!)

An example of the syntax used for this command is:

RESUME(X); /* where X would be a task number */

The task issuing this command, would then be placed in the waiting queue, for one tick time.

DEFER( )

This system command is used by a task in order to hand over processor time to another task. The task is simply placed in the Waiting Queue, actually waiting for just 1 tick, while a new task resumes execution.

This system command performs the following:

- It sets its timeout period in the TTS table to 1. The task will therefore be ready to execute after the next tick.
- It then flows on to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

```
DEFER();
/* the task simply stops execution and is placed in the Waiting Queue.*/
/* A new task would then take over. */
```

Variables Memory Map

Table B-5 shows the way the variables used in this RTOS program have been set up. Most of the variables reside in the internal 256 RAM of the 8032 micro-processor. The external RAM (from address 8100H and higher for the Flight 32 board) is used to store the stacks of all the tasks and main idle program. These stacks are then swapped in turn with the area reserved for the stack in the internal RAM whenever a task swap is necessary.

Label	**Hex Byte Address**	**Remarks Hex bit address**								**Notes**
	FF To 80	Indirect General Purpose RAM (80 – FF) which can Be used as a Stack Area								
MAIN_STACK	7F to 76	Direct and Indirect RAM (00 – 7F)								
SP (initially) T_SLOT_RELOAD	75 to 61	(NOOFTSKS+1) bytes								Time slot Reload values For each task
T_SLOT	60 to 4C	(NOOFTSKS+1) bytes								Time slot Counter For each task
SPTS	4B to 37	(NOOFTSKS+1) bytes								Storage area For the SPs Of each task
READYQ	35 to 22	(NOOFTSKS+1) bytes								Queue for Tasks ready To run
	21	0F	0E	0D	0C	0B	0A	09	08	Spare bits
	20	07	06	05	04	03	02	01	00	MYBITS
	1F to 17									Storage for any Applications variables
TMPSTORE0	16									See FETCH_STACK
GOPARAM	15									RTOSGOMSEC
DELAYHI	14									RTOSGOMSEC
DELAYLO	13									RTOSGOMSEC
TICKCOUNT	12									RTOSGOMSEC
RUNNING	11	Currently running task								Task number
READYQTOP	10	Points to last task in READYQ								Pointer
	0F to 08	Register Bank 1 (R0 – R7)								Register bank Used by the RTOS

Label	Hex Byte Address	Remarks Hex bit address	Notes
	07 to 00	Register Bank 0 (R0 - R7)	Register bank used by ALL tasks

Table B-5 PaulOS.A51 Variables setup, with 18 (12H) tasks. (NOOFTSKS=12H)

The program listing for the assembly code version of PaulOS RTOS now follows. It consists of:

- The header file PaulOS.h
- The startup file Startup.a51
- The main source file PaulOS.a51

PaulOS.h

```
/* PaulosV5C.h */
/* for use with PaulosV5C.a51 RTOS program */
/* written by Paul P. Debono - FEBRUARY 2005 */

#define uchar unsigned char
#define uint unsigned int
#define ulong unsigned long

// The following calls do not receive parameters, hence are not
// declared with an underscore prefix in the a51 file

void SET_IDLE_MODE(void);
void SET_POWER_DOWN(void);
void DEFER(void);
void KILL(void);
uchar SCHEK(void);
uchar RUNNING_TASK_ID(void);
void WAITV(void);

// The following calls do receive parameters, hence are declared
// with an underscore prefix in the a51 file

void INIT_RTOS(uchar IEMASK);
void RTOSGOMSEC(uchar msecs,uchar prior);
void SIGNAL(uchar task);
void WAITI(uchar intnum);
void WAITT(uint ticks);
void WAITS(uint ticks);
void CREATE(uchar task,uint *taskadd);
void PERIODIC(uint ticks);
void RESUME(uchar task);

/* ====================================================      */
/* ADD-ON MACROS */
/* ====================================================      */
```

```
/* Macro 'WAITM' Used to wait for a max of 1 MINUTE                 */
/* Use with an RTOSGOMSEC(1,0) - 1 milli seconds tick time          */
#define WAITM(s, ms) WAITT((uint)(##s*1000 +##ms))

/* Macro 'WAITH' Used to wait for a max of 18h 12m                  */
/* 66535 * 4 * 250 MILLISECONDS                                     */
/* Use with an RTOSGOMSEC(250,0) - 250 milli seconds tick time      */
#define WAITH(H, M, S)  {WAITT((uint)(3600*##H + 60*##M + ##S));     \
                      WAITT((uint)(3600*##H + 60*##M + ##S));       \
                      WAITT((uint)(3600*##H + 60*##M + ##S));       \
                      WAITT((uint)(3600*##H + 60*##M + ##S));       \
                                              }
/* Macro 'WAITD' Used to wait for a max of 7D 14h 2m                */
/* 66535 * 40 * 250 MILLISECONDS                                    */
/* Use with an RTOSGOMSEC(250,0) - 250 milli seconds tick time      */
#define WAITD(D, H, M)  {WAITT((uint)(8640*##D + 360*##H + 6*##M));  \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
                      WAITT((uint)(8640*##D + 360*##H + 6*##M));    \
```

```
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                WAITT((uint)(8640*##D + 360*##H + 6*##M));   \
                }

/* Macro 'PERIODICA' Used to wait for a max of 4h 33m              */
/* Use with an RTOSGOMSEC(250,0) - 250 milli seconds tick time     */
#define PERIODICA(H, M, S) PERIODIC((uint)(14000*##H + 240*##M + 4*##S))

/* Macro 'PERIODICM' Used to wait for a max of 1 MINUTE            */
/* Use with an RTOSGOMSEC(1,0) - 1 milli seconds tick time         */
#define PERIODICM(s, ms) PERIODIC((uint)(##s*1000 +##ms))

/* ========================================================        */
```

TaskStk.a51

```
;TASKSTKV5C.a51
$NOMOD51
#include "..\Headers\reg52.h"
; check your own correct path
USING 1        ; SET ASIDE BANK 1
;
; ****************************************************************************
;                              I M P O R T A N T
INT_VECTOR_BASE EQU 7FFDH            ; INTERRUPT VECTOR TABLE BASE ADDRESS
TICK_TIMER EQU 2                     ; SELECT WHICH TIMER TO USE FOR RTOS TICKS
USING_INT EQU 0                      ; SET TO 1 IF USING INTERRUPTS (WAITI)
PERIODIC_CMD EQU 0                   ; IF NOT USING PERIODIC COMMAND SET TO ZER0
                                     ; TO CLEAR UP SOME INTERNAL IDATA MEMORY
HALFMSEC EQU 1                       ; SET TO 1 TO CHECK INTERRUPTS EVERY 1/2 MSEC
```

```
                                        ; ELSE RTOS WOULD CHECK EVERY 1 MSEC

NOOFTSKS EQU 7                          ; CAN BE MORE, SAY 20 TASKS (numbered 0 to 19)
MAIN_STACK EQU 0BFH                     ; CONFIRM LOCATION WITH KEIL FILE *.M51
                                        ; see variable ?STACK in IDATA
;
STACKSIZE EQU 25H                       ; 20H MINIMUM
;
; NOTE ************************************************************************
;       MODIFY ABOVE TO REFLECT YOUR APPLICATION PROGRAM AND HARDWARE
; *****************************************************************************
;
; THESE ARE THE FOUR MAIN PARAMETERS WHICH YOU MIGHT NEED TO ADJUST,
; DEPENDING ON YOUR APPLICATION.
;
; A STACK SIZE OF 20H SHOULD BE ADEQUATE FOR MOST APPLICATIONS.
;
; ****************************************************************************
#include "..\PaulosRTOS\RTMACROSV5C.a51"
#include "..\PaulosRTOS\PaulosV5C.a51"
; ****************************************************************************
```

RTMacros.a51

```
;
; RTMACROSV5C.A51
;
; RTOS EQUATES
; FOR USE WITH PAULOSV5C.A51 RTOS.
;
       EXT0_INT_VECTOR EQU (INT_VECTOR_BASE + 03H)
       TIM0_INT_VECTOR EQU (INT_VECTOR_BASE + 0BH)
       EXT1_INT_VECTOR EQU (INT_VECTOR_BASE + 13H)
       TIM1_INT_VECTOR EQU (INT_VECTOR_BASE + 1BH)
       SER0_INT_VECTOR EQU (INT_VECTOR_BASE + 23H)
       TIM2_INT_VECTOR EQU (INT_VECTOR_BASE + 2BH)

;
IF (HALFMSEC = 1)
       RTCLOCK        EQU 461           ; timer clock (11059/12 = 922) counts for 1 msec
ELSE                                    ; assuming 11.0592 MHz crystal
       RTCLOCK        EQU 922
ENDIF

BASIC_TICK EQU (65535 - RTCLOCK + 1)

NOOFPUSHES EQU 13                       ; Number of pushes at beginning of Task change
                                        ; i.e. pushes in PushBank0.
IDLE_TASK EQU NOOFTSKS                  ; main endless loop in C application given
                                        ; a task number equal to NOOFTSKS
NOT_TIMING EQU 0H

; TASK FLAG MASKS
SIGS           EQU 10000000B            ; 128
SIGW           EQU 01000000B            ; 64
```

```
SIGV            EQU 00100000B          ; 32
SER0W           EQU 00010000B          ; 16
EXT1W           EQU 00000100B          ; 4
EXT0W           EQU 00000001B          ; 1

IF (TICK_TIMER = 2)
  TIM0W         EQU 00000010B          ; 2
  TIM1W         EQU 00001000B          ; 8
ELSEIF (TICK_TIMER = 1)
  TIM0W         EQU 00000010B          ; 2
  TIM2W         EQU 00001000B          ; 8
ELSEIF (TICK_TIMER = 0)
  TIM1W         EQU 00000010B          ; 2
  TIM2W         EQU 00001000B          ; 8
ENDIF

; ========================================================

; ========================================================

; RTOS MACROS
;

SetBank MACRO BankNumber
IF BankNumber = 0
  CLR RS0
  CLR RS1
ELSEIF BankNumber = 1
  SETB RS0
  CLR RS1
ELSEIF BankNumber = 2
  SETB RS1
  CLR RS0
ELSEIF BankNumber = 3
  SETB RS1
  SETB RS0
ENDIF
ENDM

Ext2Int MACRO          ; MOVES R0 DATA FROM EXT DPTR POINTER TO INTERNAL R1 POINTER
  MOV R1,#MAIN_STACK
  MOV R0,#STACKSIZE
NEXT11:
  MOVX A,@DPTR
  MOV @R1,A
  INC DPTR
  INC R1
  DJNZ R0,NEXT11
ENDM

Int2Ext MACRO          ; MOVES R0 DATA FROM INTERNAL R1 POINTER TO EXT DPTR PONTER
                       ; USES R0, R1, ACC AND DPTR
  MOV R1,#MAIN_STACK
  MOV R0,#STACKSIZE
NEXT12:
  MOV A,@R1
```

```
  MOVX @DPTR,A
  INC DPTR
  INC R1
  DJNZ R0,NEXT12
ENDM

Push_Bank0_Reg MACRO                 ; 13 PUSHES
  PUSH ACC
  PUSH B
  PUSH PSW
  PUSH DPL
  PUSH DPH
  PUSH 00
  PUSH 01
  PUSH 02
  PUSH 03
  PUSH 04
  PUSH 05
  PUSH 06
  PUSH 07
ENDM

Pop_Bank0_Reg MACRO
  POP 07
  POP 06
  POP 05
  POP 04
  POP 03
```

```
  POP 02
  POP 01
  POP 00
  POP DPH
  POP DPL
  POP PSW
  POP B
  POP ACC
ENDM

Push_HalfB0_Reg MACRO         ; R7 NOT PUSHED, USED FOR PASSING PARAMETER
 PUSH ACC                     ; BACK TO MAIN CALLING PROGRAM
 PUSH B
 PUSH PSW
 PUSH DPL
 PUSH DPH
 PUSH 00
 PUSH 01
 PUSH 02
 PUSH 03
ENDM

Pop_HalfB0_Reg MACRO          ; R7 NOT POPPED, USED FOR PASSING PARAMETER
  POP 03                      ; BACK TO MAIN CALLING PROGRAM
  POP 02
  POP 01
  POP 00
  POP DPH
  POP DPL
  POP PSW
  POP B
  POP ACC
ENDM

DEC2REGS MACRO LowReg, HighReg
LOCAL HIGHOK
       CLR     C                     ; Clear For SUBB
       MOV     A,LowReg              ; Move Low Of DPTR To A
       SUBB    A,#1                  ; Subtract 1
       MOV     LowReg,A              ; Store Back
       JNC     HIGHOK
       MOV     A,HighReg             ; Get High Of DPTR
       SUBB    A,#0                  ; Subtract CY If Set
       MOV     HighReg,A             ; Move Back
HIGHOK:
ENDM

LOADREGSXDATA MACRO LowReg, HighReg
       MOVX A,@DPTR
       MOV LowReg,A
       INC DPTR
       MOVX A,@DPTR
       MOV HighReg,A
ENDM
```

```
LOADXDATAREGS MACRO LowReg, HighReg
       MOV A,LowReg
       MOVX @DPTR,A
       INC DPTR
       MOV A,HighReg
       MOVX @DPTR,A
ENDM

DPTRPLUSA MACRO
       ADD A,DPL                    ; Add 'A' To DPL
       MOV DPL,A                    ; Move Result To DPL
       MOV A,DPH                    ; Get DPH
       ADDC A,#0                    ; If Carry Set, This Will Increment
       MOV DPH,A                    ; Move Back To DPH
ENDM
; ***********************************************
```

PaulOS.A51

```
; ***********************************************
; PaulOSV5C.A51 FOR C USE
; Version 5C
; ****************************************************
;
; STORES ALL BANK 0 TASK REGISTERS
;
; NOTE THAT MAIN_STACK WOULD HAVE TO BE VERIFIED
; WITH FILE *.M51
;
; HANDLES MANY TASKS, DEPENDING ON
; EXTERNAL MEMORY AND INTERNAL STACK SPACE
; CAN BE USED WITH ASSEMBLY LANGUAGE MAIN PROGRAM
;
; Written by Paul P. Debono - FEBRUARY 2005
; University of Malta
; Department of Communications and Computer Engineering
; MSIDA MSD 06; MALTA.
;
;
;/********************************************************************************/
;//
;//            N O T E
;//     USE the following settings in Options for Target 1
;// Memory Model: LARGE: VARIABLES IN XDATA
;// Code Model: LARGE: 64K Program
;//            START          SIZE           (32K RAM)
;// CODE:      0X8100         0X5D00
;// RAM:       0XDE00         0X2000
;//     or say
;// CODE:      0X8100         0X4000
;// RAM:       0XC100         0X3D00
;//
;// CODE:      0X8100         0X1B00         (8K RAM)
;// RAM:       0X9C00         0X400
```

```
;//
;// Code Model:        LARGE: 64K Program
;//             START          SIZE           (32K EPROM)
;// CODE:       0X0000         0X8000
;// RAM:        0X8000         0X7E00
;//
;/*******************************************************************************/

; STACK MOVING VERSION - MOVES WORKING STACK IN AND OUT OF
; EXTERNAL MEMORY
; SLOWS DOWN RTOS, BUT DOES NOT RESTRICT TASK CALLS
;
; Uses timer 2, in 16-bit auto-reload mode as the time scheduler (time-ticker)
; FOR 8051, TIMER 0 CAN BE USED.
; All tasks run in bank 0, RTOS kernel runs in bank 1
; All tasks must be written as an endless loop.
;
; Waiting time range for WAITT system calls is 1-65535.
; A zero waiting time parameter is set to 1 by the RTOS itself,
; since a ZERO effectively kills the task,
; actually putting it in permanent sleep in the waiting queue!!
;
; Waiting time range for WAITS system call is 0-65535. 0 means wait for the signal
; forever
;
; IDLE TASK (ENDLESS MAIN PROGRAM - TASK NUMBER = NOOFTASKS)
;
```

```
; COMMANDS AVAILABLE FOR THE C APPLICATION PROGRAM ARE:
; (valid parameter values are shown in parenthesis, assuming 20 tasks maximum)
; THE TOP FIVE COMMANDS ARE USED ONLY IN THE MAIN (IDLE TASK) PROGRAM.
; THE OTHERS ARE ONLY USED IN THE TASKS.
;
;                      THE FOLLOWING COMMANDS, DO NOT CAUSE A CHANGE OF TASK:
; INIT_RTOS(IEMASK)    Initialise variables, SPs and enable required interrupts.
;                      ***** THIS MUST BE THE FIRST RTOS COMMAND TO BE EXECUTED *****
; VALID INTERRUPT NUMBERS USING OLD MONITOR ARE 0, 2, 3 AND 4
; VALID INTERRUPT NUMBERS USING NEW MONITOR OR USER EEPROM ARE 0, 1, 2, 3 AND 4
; NOTE THAT IF TIMER 1 IS BEING USED TO GENERATE BAUD RATE,
; THEN YOU CANNOT USE 3 AND 4 SIMULTANEOUSLY

; 0     EXTERNAL INT   0       (IEMASK = 00000001 = 01H)
; 1     TIMER COUNTER  0       (IEMASK = 00000100 = 02H)
; 2     EXTERNAL INT   1       (IEMASK = 00000100 = 04H)
; 3     TIMER COUNTER  1       (IEMASK = 00001000 = 08H)
; 4     SERIAL PORT            (IEMASK = 00010000 = 10H)
; 5     TIMER COUNTER  2       (IEMASK = 00100000 = 20H)
;
;
;
; CREATE(TSK#,TSKADDR) Create a new task (0-[n-1]),placing it in the Ready Queue,
;                              and set up correct task address on its stack.
; RTOSGOMSEC(TICKTIME,PRIORITY)
;                       Start RTOS going, interrupt every TICKTIME (1-255) msecs.
;                       PRIORITY = 1 implies Q Priority sorting is required.
;                       PRIORITY = 0 implies FIFO queue function.
;
; SET_IDLE_MODE()      Puts micro-controller in Idle mode           (IDL, bit 0 in PCON)
; SET_POWER_DOWN()     Puts micro-controller in Power Down mode     (PD, bit 1 in PCON)
;
; PERIODIC(TIME)       Repeat task every TIME msecs.
; SCHEK()              Check if current task has its signal set (Returns 1 or 0).
;                              Signal is cleared if it was found to be set.
; SIGNAL(TASKNUMBER)   Set signal bit of specified task (0-[n-1]).
; RUNNING_TASK_ID()    Returns the number of the currently executing task
;
;                      THE FOLLOWING COMMANDS WILL CAUSE A CHANGE IN TASK ONLY
;                              WHEN THE SIGNAL IS NOT ALREADY PRESENT.
;
; WAITS(TIMEOUT)       Wait for signal within TIMEOUT ticks (TIMEOUT = 1 - 65535).
;                              Or wait for signal indefinitely (TIMEOUT = 0).
;                              If signal already present, proceed with current task.
; WAITV()              Wait for interval to pass.
;                              If interval already passed, proceed with current task.
;
;                      THE FOLLOWING COMMANDS, ALWAYS CAUSE A CHANGE IN TASK:
;
; WAITT(TIMEOUT)       Wait for timeout ticks (1 - 65535).
; WAITI(INTNUM)        Wait for the given interrupt to occur.
; DEFER()              Stop current task and let it wait for 1 tick.
; KILL()               Kill current task by marking it permanently waiting,
```

```
;                             (TIMEOUT = 0). Clears any waiting signals.
; RESUME(TASKNUMBER)   Resumes the requested task which had previously been KILLed
;
;
; THIS IS STILL A SMALL TEST VERSION RTOS. IT IS JUST USED FOR
; SHOWING WHAT IS NEEDED TO MAKE A SIMPLE RTOS.
; IT MIGHT STILL NEED SOME MORE FINE TUNING.
; IT HAS NOT BEEN THOROUGHLY TESTED YET !!!!
; BUT WORKS FINE SO FAR.
; NO RESPONSABILITY IS TAKEN.
;
; CHECK YOUR OWN CORRECT FILE NAME INCLUDING CORRECT PATH IF NECESSARY.
;
; NOTE: Functions which receive parameters when
;       called from within C must have their name
;       start with an underscore in the A51 source file.
;
;
; These two parameters (set in TaskStkV5C.A51) are used to save
; code and data memory space and increase rtos performance if these
; functions are not being used.
IF (USING_INT = 1)
       PUBLIC _WAITI
ENDIF

IF (PERIODIC_CMD = 1)
       PUBLIC WAITV, _PERIODIC
ENDIF

PUBLIC DEFER, KILL, SCHEK                     ; no parameters
PUBLIC SET_IDLE_MODE, SET_POWER_DOWN          ; no parameters
PUBLIC RUNNING_TASK_ID                        ; no parameters

PUBLIC _INIT_RTOS, _CREATE
PUBLIC _WAITT, _WAITS
PUBLIC _RTOSGOMSEC, _SIGNAL, _RESUME

; CHECK YOUR OWN CORRECT FILE NAME INCLUDING CORRECT PATH IF NECESSARY.

RTOSVAR1 SEGMENT DATA
RSEG RTOSVAR1                 ; VARIABLE DATA AREA VAR1,
                              ; range 0x10-0xFF, since we are using Banks 0,1
      READYQTOP:  DS 1        ; ADDRESS OF LAST READY TASK
      RUNNING:    DS 1        ; NUMBER OF CURRENT TASK
      TMPSTORE0:  DS 1        ; USED IN FETCHSTACK
      XINTMASK:   DS 1        ; MASK SET BY EXTERNAL INTERRUPT TO INDICATE TYPE
      TICKCOUNT:  DS 1        ; USED FOR RTOSGO.....
      GOPARAM:    DS 1        ; USED FOR RTOSGO.....

MYRTOSBITS SEGMENT BIT
RSEG MYRTOSBITS
IF (HALFMSEC = 1)
       MSECFLAG: DBIT 1              ; FLAG INDICATING 1 MSEC PASSED
ENDIF
       INTFLAG: DBIT 1               ; MARKER INDICATING FOUND TASK WAITING FOR
                                     ; SOMEINTERUPT
       TINQFLAG: DBIT 1              ; TASK TIMED OUT MARKER
```

```
        PRIORITY: DBIT 1               ; PRIORITY BIT SET BY RTOSGO....

RSEG RTOSVAR1 ; DIRECTLY ADDRESSABLE AREA
        READYQ: DS (NOOFTSKS + 2) ; QUEUE STACK FOR TASKS READY TO RUN

; THE FOLLOWING VARIABLES CAN BE IN THE INDIRECTLY ADDRESSABLE RAM (EVEN > 80H)
RTOSVAR2 SEGMENT IDATA
RSEG RTOSVAR2
        SPTS:    DS (NOOFTSKS + 1)     ; SP FOR EACH TASK AND 1 FOR THE IDLE (MAIN) TASK
        TTS:     DS 2*NOOFTSKS         ; REMAINING TIMEOUT TIME FOR TASKS, 2 BYTES PER TASK
                                       ; 0 = NOT TIMING
        TSKFLAGS: DS (NOOFTSKS + 1)    ; BYTES STORING FLAGS FOR EACH TASK (AND MAIN)

; MAIN_STACK AREA STARTS HERE, NEXT LOCATION AFTER TSKFLAGS.
; CHECK STACK LOCATION IN THE .M51 FILE AFTER COMPILING
; TO CONFIRM THE VALUE OF "MAIN_STACK"

;XSEG AT (XTRAMTOP - (STACKSIZE * (NOOFTSKS + 1)) - (4*NOOFTSKS) + 1)
EXTERNDATA SEGMENT XDATA
RSEG EXTERNDATA

IF (PERIODIC_CMD = 1)
        INTVALCNT: DS 2*NOOFTSKS        ; 0 = NOT TIMING
        INTVALRLD: DS 2*NOOFTSKS        ; 0 = NOT TIMING
ENDIF
        EXT_STK_AREA: DS (NOOFTSKS + 1) * STACKSIZE ; THIS IS THE ACTUAL SIZE OF STACK AREA
; ======================================================================
```

```
;

CSEG AT EXT0_INT_VECTOR ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA
       MOV XINTMASK,#EXT0W           ; EXTERNAL 0
       LJMP XTRA_INT

IF (TICK_TIMER = 0)
CSEG AT TIM0_INT_VECTOR              ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA                        ; TIMER 0
       LJMP RTOS_TIMER_INT           ; USED FOR THE RTOS SCHEDULER
ELSE
CSEG AT TIM0_INT_VECTOR
       CLR EA
       MOV XINTMASK,#TIM0W
       LJMP XTRA_INT
ENDIF

CSEG AT EXT1_INT_VECTOR              ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA
       MOV XINTMASK,#EXT1W           ; EXTERNAL 1
       LJMP XTRA_INT

IF (TICK_TIMER = 1)
CSEG AT TIM1_INT_VECTOR ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA                        ; TIMER 1
       LJMP RTOS_TIMER_INT           ; USED FOR THE RTOS SCHEDULER
ELSE
CSEG AT TIM1_INT_VECTOR
       CLR EA
       MOV XINTMASK,#TIM1W
       LJMP XTRA_INT
ENDIF

CSEG AT SER0_INT_VECTOR              ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA
        MOV XINTMASK,#SER0W          ; SERIAL
        LJMP XTRA_INT

IF (TICK_TIMER = 2)
CSEG AT TIM2_INT_VECTOR              ; INTERRUPT VECTOR ADDRESS FOR
       CLR EA                        ; TIMER 2
       CLR TF2                       ; Clear Timer 2 interrupt flag (not done automatically)
       LJMP RTOS_TIMER_INT
ELSE
CSEG AT TIM2_INT_VECTOR
       CLR EA
       MOV XINTMASK,#TIM2W
       LJMP XTRA_INT

ENDIF
```

```
MyRTOS_CODE SEGMENT CODE             ; STARTS AT 8100H FOR THE FLIGHT32 BOARD
RSEG MyRTOS_CODE
;==========================================================================
; START OF RTOS SYSTEM
; PREFIX NAME FOR FUNC WITH REG-PASSED PARAMS MUST START WITH AN UNDERSCORE _

SET_IDLE_MODE:                       ; SETS THE MICRO-CONTROLLER IN IDLE MODE
       ORL PCON,#0x01                ; SETS BIT 0 OF PCON SFR
       RET

SET_POWER_DOWN:                      ; SETS THE MICRO-CONTROLLER IN POWER DOWN MODE
       ORL PCON,#0x02                ; SETS BIT 1 OF PCON SFR
       RET

_INIT_RTOS:                          ; SYS CALL TO SET UP VARIABLES
                                     ; R7 HOLDS THE IE MASK
       MOV A,R7
       ANL A,#01111111B              ; ENSURE EA = 0 (ENABLED LATER FROM RTOSGO...)
 IF (TICK_TIMER = 0)
       ORL A,#00000010B              ; AND ET0 = 1 (USED FOR RTOS TICK TIME)
       MOV IP,#02H                   ; Timer 0 High Priority, PT0=1, OTHERS ALL LOW
 ELSEIF (TICK_TIMER = 1)
       ORL A,#00001000B              ; AND ET1 = 1 (USED FOR RTOS TICK TIME)
       MOV IP,#08H                   ; Timer 1 High Priority, PT1=1, OTHERS ALL LOW
 ELSEIF (TICK_TIMER = 2)
       ORL A,#00100000B              ; AND ET2 = 1 (USED FOR RTOS TICK TIME)
       MOV IP,#20H                   ; Timer 2 High Priority, PT2=1, OTHERS ALL LOW
 ENDIF

       MOV IE,A
; IN THE C ENVIRONMENT, THE KEIL SOFTWARE CLEARS THE INTERNAL RAM FROM 0 TO FFH
; PROVIDED THAT THE C51\LIB FILE STARTUP.A51 IS INCLUDED WITH THE SOURCE GROUP,
; AND WITH THE CORRECT IDATALEN VARIABLE SETTING TO REFLECT 8051 FAMILY TYPE.
;
; IN ASM OR A51 (NOT IN C), ALL THE INTERNAL RAM (0-FFH) IS
; CLEARED BY MEANS OF THE CLR_8051_RAM MACRO.
; IN C it is cleared when using STARTUP.A51
;

; CLEAR PERIODIC INTERVAL TABLE IF BEING USED
IF (PERIODIC_CMD = 1)
       MOV DPTR,#INTVALCNT
       MOV A,#NOOFTSKS
       RL A                          ; DOUBLE THE NUMBER
       MOV R0,A                      ; R0 CONTAINS NUMBER OF BYTES TO CLEAR
       CLR A
DPTR_A0: ; POINT DPTR TO CORRECT LOCATION
       MOVX @DPTR,A
       INC DPTR
       DJNZ R0,DPTR_A0
ENDIF

       MOV DPTR,#EXT_STK_AREA        ; CLEAR ALL EXTERNAL RAM STACKS
       MOV R0,#(NOOFTSKS + 1)
       CLR A
```

```
NEXT_STACK:
        MOV R1,#STACKSIZE
CLR_STACK:
        MOVX @DPTR,A
        INC DPTR
        DJNZ R1,CLR_STACK
        DJNZ R0,NEXT_STACK
        MOV RUNNING,#IDLE_TASK        ; IDLE TASK RUNNING (Main program endless loop)
        MOV R7,#NOOFTSKS
        MOV R0,#TTS
        MOV R1,#READYQ
LOAD_VARS:
        MOV @R0,#LOW(NOT_TIMING)      ; NO TIMER ACTION
        INC R0
        MOV @R0,#HIGH(NOT_TIMING)
        MOV @R1,#IDLE_TASK    ; IDLE TASK IN ALL OF READYQ (Main program endless loop)
        INC R0
        INC R1
        DJNZ R7,LOAD_VARS             ; SET UP ALL TASKS
        MOV @R1,#IDLE_TASK            ; FILL TWO ADDITIONAL LOCATIONS, USED
        INC R1                        ; DURING THE Q SHIFTING ROUTINE, WITH IDLE TASK.
        MOV @R1,#IDLE_TASK            ; THIS ENSURES IDLE TASK WILL ALWAYS BE IN Q IF
        MOV READYQTOP,#READYQ         ; THERE ARE NO OTHER TAKS READY TO EXECUTE.
                                              ; SET UP SP
        MOV R7,#(NOOFTSKS + 1)                ; COUNTER
        MOV R0,#SPTS                          ; INITIALIZE ALL STACK POINTERS
        MOV A, #(MAIN_STACK - 1)
        ADD A,#(NOOFPUSHES + 2)               ; SIMULATE Push_Bank0_Reg PLUS
                                              ; SAVING OF RETURN ADDRESS BY INTERRUPT
SET_UP:
        MOV @R0,A
        INC R0
        DJNZ R7,SET_UP
        RET

_CREATE:
                                              ; SYS CALL ENTRY TO CREATE A TASK
                                              ; TASK NUMBER (0 to 19) PASSED IN BANK0 R7
                                              ; TASK START ADDR PASSED IN BANK0 R1,R2,R3
                                              ; LSB in R1, MSB in R2, R3 contains type
        INC READYQTOP                         ; POINT TO TOP OF READY READYQ
        MOV R0,READYQTOP
        MOV @R0,07H                           ; PUT TASK# (R7 bank 0 = 07H) IN READY QUEUE
        MOV A,R7
        CALL FETCH_STACK
        MOV A,R1
        MOVX @DPTR,A                          ; COPY LOW BYTE R1 INTO LOW STACK AREA
        INC DPTR
        MOV A,R2
        MOVX @DPTR,A                          ; NOW SAVE THE HIGH ORDER BYTE (R2)
        SETB TINQFLAG                         ; SIGNAL NEW TASK IN Q, USED TO START QSHIFT
        RET

_RTOSGOMSEC:                  ; SYS CALL TO START RTOS FOR R7 MILLISECOND TICKS
```

```
        CLR PRIORITY
        CJNE R5,#1,PRIORITY_OK              ; IF SECOND PARAMETER = 1, THEN
        SETB PRIORITY                       ; SET PRIORITY SORTING IS REQUIRED
PRIORITY_OK:
 IF (TICK_TIMER = 0)
        MOV TH0,#HIGH(BASIC_TICK)           ; LOAD TH0 AND TL0 WITH BASIC TICK COUNT
        MOV TL0,#LOW(BASIC_TICK)            ; SAVE THEM IN THE AUTO RE-LOAD REGISTERS
 ELSEIF (TICK_TIMER = 1)
        MOV TH1,#HIGH(BASIC_TICK)           ; LOAD TH0 AND TL0 WITH BASIC TICK COUNT
        MOV TL1,#LOW(BASIC_TICK)            ; SAVE THEM IN THE AUTO RE-LOAD REGISTERS
 ELSE
        MOV RCAP2H,#HIGH(BASIC_TICK)        ; LOAD RCAPS WITH 1 MILLISECOND COUNT
        MOV RCAP2L,#LOW(BASIC_TICK)         ; SAVE THEM IN THE AUTO RE-LOAD REGISTERS
 ENDIF ; OF TIMER 2 (FOR FLT-32)

        MOV GOPARAM,07              ; LOAD TICKS PARAMETER, PASSED IN R7 BANK 0
        MOV TICKCOUNT,07
 IF (TICK_TIMER = 0)
        ANL TMOD,#0F0H
        ORL TMOD,#01H               ; START TIMER 0 IN 16-BIT MODE 1.
        SETB TF0                    ; SIMULATE TIMER 0 INTERRUPT.
 ELSEIF (TICK_TIMER = 1)
        ANL TMOD,#0FH
        ORL TMOD,#10H               ; START TIMER 1 IN 16-BIT MODE 1.
        SETB TF1                    ; SIMULATE TIMER 1 INTERRUPT.
 ELSEIF (TICK_TIMER = 2)
        MOV T2CON,#04H              ; START TIMER 2 IN 16-BIT AUTO RE-LOAD MODE.
```

```
                                      ; TIMER 1 CAN BE USED FOR SERIAL BAUD RATE
        SETB TF2                      ; SIMULATE TIMER 2 INTERRUPT IMMEDIATELY
 ENDIF
        SETB EA                       ; ENABLE GLOBAL INTERRUPT SIGNAL
        RET                           ; EFFECTIVELY STARTING THE RTOS.

SCHEK:                                ; SYS CALL ENTRY CHECK SIGNAL BIT FOR TASK
                                      ; RETURN 0 IF BIT CLEAR OR 1 IF BIT SET IN R7.
                                      ; SIG. BIT IS CLEARED IF FOUND TO BE SET
                                      ; NO NEED FOR BANK SWITCHING
        CLR EA                        ; IMMEDIATE RETURN - NO CONTEXT SWITCHING
        Push_HalfB0_Reg
        MOV A,RUNNING
        MOV B,#SIGS
        CALL CHK_CLR_FLAG             ; SIG IS CLEARED IF IT WAS FOUND TO BE SET
        MOV R7,#1
        JC SIGNAL_SET                 ; SIG SET, HENCE RETURN WITH R7=1
        DEC R7                        ; SIG NOT YET SET, HENCE RETURN WITH R7=0
SIGNAL_SET:
        Pop_HalfB0_Reg
        SETB EA
        RET

_SIGNAL:                              ; SYS CALL ENTRY-SET SIGNAL BIT FOR SPECIFIED TASK
        CLR EA                        ; NO NEED FOR BANK SWITCHING - NO CONTEXT SWITCHING
        Push_Bank0_Reg
        MOV A,R7                      ; TASK NUMBER PASSED IN R7 bank 0
        MOV B,#SIGW
        CALL CHK_CLR_FLAG
        JNC NOT_WAITING               ; IF TASK NOT ALREADY WAITING, SET SIGNAL BIT
        MOV A,R7                      ; OTHERWISE PLACE IT ON READY Q
        MOV B,#SIGS                   ; ENSURE CLEARED SIGNAL BIT
        CALL CLR_FLAG
        MOV A,#TTS                    ; AND MARK TASK AS NOT TIMING
        ADD A,R7
        ADD A,R7                      ; ADD OFFSET TWICE SINCE 2 TIME-OUT BYTES
                                      ; PER TASK
        MOV R0,A
        MOV @R0,#LOW(NOT_TIMING)
        INC R0
        MOV @R0,#HIGH(NOT_TIMING)
        INC READYQTOP
        MOV R0,READYQTOP
        MOV @R0,07                    ; PLACE SIGNALLED TASK ON READY Q
        SETB TINQFLAG                 ; INDICATE, NEW TASK IN Q, BUT
DONT_GIVE_UP:
        Pop_Bank0_Reg
        SETB EA                       ; DON'T GIVE UP RUNNING CURRENT TASK.
        RET                           ; (MUST DEFER IF REQUIRED TO DO SO)
NOT_WAITING:
        MOV A,R7
        MOV B,#SIGS                   ; SET SIGNAL BIT OF SIGNALLED TASK
        CALL SET_FLAG
        Pop_Bank0_Reg
        SETB EA
        RET                           ; AND CONTINUE RUNNING CURRENT TASK
```

```
IF (USING_INT = 1)
_WAITI:                                  ; SYS CALL ENTRY POINT - WAIT FOR INTERRUPT
; VALID INTERRUPT NUMBERS USING MONITOR ARE 0, 2, 3 AND 4
; VALID INTERRUPT NUMBERS USING USER EEPROM ARE 0, 1, 2, 3 AND 4
; NOTE THAT IF TIMER 1 IS BEING USED TO GENERATE BAUD RATE,
; THEN YOU CANNOT USE 3 AND 4 SIMULTANEOUSLY
; INT 5 IS COMPULSORY         (IEMASK = 00100000 = 20H)
; 0     EXTERNAL INT 0        (IEMASK = 00000001 = 01H)
; 1     TIMER COUNTER 0       (IEMASK = 00000010 = 02H)
; 2     EXTERNAL INT 1        (IEMASK = 00000100 = 04H)
; 3     TIMER COUNTER 1       (IEMASK = 00001000 = 08H)
; 4     SERIAL PORT           (IEMASK = 00010000 = 10H)
; 5     TIMER COUNTER 2       (IEMASK = 00100000 = 20H)

        CLR EA
        Push_Bank0_Reg
        INC R7                ; INTERRUPT NUMBER (0 TO 4) PARAMETER PASSED IN R7 bank 0
        CLR A
        SETB C
SHIFT_LEFT:                              ; CONVERT TO INTERRUPT MASK (1,2,4,8,16) BY ROTATING LEFT
        RLC A
        DJNZ R7, SHIFT_LEFT
        MOV B,A                          ; B NOW CONTAINS CORRECT INTERRUPT MASK
        MOV A,RUNNING
        CALL SET_FLAG
        LJMP QSHFT                       ; STOP CURRENT TASK AND RUN NEXT TASK IN READY Q
ENDIF

IF (PERIODIC_CMD = 1)
WAITV:
        CLR EA                           ; UNTIL TIMEOUT PASSED IN R7(LOW),R6(HIGH)
        Push_Bank0_Reg
        MOV A,RUNNING
        MOV B,#SIGV                      ; TEST IF SIGNAL ALREADY THERE
        CALL CHK_CLR_FLAG
        JNC NO_INTVAL                    ; NO SIGNAL YET, SO TASK MUST WAIT
                                         ; ELSE, SIGNAL WAS PRESENT, (NOW CLEARED)
        LJMP DONT_GIVE_UP                ; OR RETURN TO SAME TASK
NO_INTVAL:
        MOV A,RUNNING                    ; RELOAD TASK NUMBER
        MOV B,#SIGV
        CALL SET_FLAG                    ; SET SIG WAITING BIT, AND
        LJMP QSHFT                       ; RUN NEXT TASK IN READY Q
ENDIF

_WAITS:                                  ; SYSTEM CALL - WAIT SIGNAL ARRIVAL
        CLR EA                           ; UNTIL TIMEOUT PASSED IN R7(LOW),R6(HIGH)
        Push_Bank0_Reg
        MOV A,RUNNING
        MOV B,#SIGS                      ; TEST IF SIGNAL ALREADY THERE
        CALL CHK_CLR_FLAG
        JNC NO_SIGNAL                    ; NO SIGNAL YET, SO TASK MUST WAIT
                                         ; ELSE, SIGNAL WAS PRESENT, (NOW CLEARED)
```

```
        LJMP DONT_GIVE_UP               ; OR RETURN TO SAME TASK
NO_SIGNAL:
        MOV A,RUNNING                   ; RELOAD TASK NUMBER
        MOV B,#SIGW
        CALL SET_FLAG                   ; SET SIG WAITING BIT, AND CONTINUE WITH WAITT
                                        ; TO WAIT FOR TIMEOUT
        CJNE R7,#LOW(NOT_TIMING),SET_TIMEOUT  ; ACCEPT A WAIT TIME OF 0
        CJNE R6,#HIGH(NOT_TIMING),SET_TIMEOUT ; ACCEPT ZERO WAIT TIME IN ORDER TO BE ABLE
                                        ; TO WAIT FOR SIGNAL INDEFINITELY
        SJMP SET_TIMEOUT_0

_WAITT:                                 ; SYS CALL ENTRY POINT - WAIT FOR TIME OUT
        CLR EA
        Push_Bank0_Reg

        CJNE R7,#LOW(NOT_TIMING),SET_TIMEOUT  ; TIME OUT PARAMETER PASSED IN R6 (HIGH)
        CJNE R6,#HIGH(NOT_TIMING),SET_TIMEOUT ; AND R7 (LOW) BANK 0
        MOV R7,#1                       ; RANGE 1-65535 (0 = PERMANENT SLEEP)
        MOV R6,#0                       ; IF BOTH ARE ZERO, REPLACE WITH A ONE
SET_TIMEOUT:
        CLR     C                       ; PERFORM 65536 - TIME OUT VALUE
        CLR     A                       ; SO THAT IN RTOS_TIMER_INT WE CAN
        SUBB    A,R7                    ; USE 'INC DPTR' EASILY TO UPDATE TIMEOUT
        MOV     R7,A
        CLR     A
        SUBB    A,R6
        MOV     R6,A
SET_TIMEOUT_0:
        MOV A,#TTS
        ADD A,RUNNING                   ; ADD OFFSET TWICE SINCE TIMEOUTS ARE
        ADD A,RUNNING                   ; TWO BYTES PER TASK
        MOV R0,A
        MOV @R0,07                      ; BANK 0 R7,R6 - TIMEOUT PUT IN TABLE (WAITING Q)
        INC R0
        MOV @R0,06
        LJMP QSHFT                      ; STOP CURRENT TASK AND RUN NEXT TASK IN READY Q

KILL:                                   ; SYS CALL ENTRY (NO PARAMETERS)
                                        ; CLEARS ALL WAITING SIGNALS FLAGS
        CLR EA
        Push_Bank0_Reg
        MOV A,RUNNING
        MOV R0,#TSKFLAGS
        ADD A,R0
        MOV R0,A
        CLR A
        MOV @R0,A                       ; TO CLEAR AND STORE
        MOV R7,#LOW(NOT_TIMING)         ; KILL PRESENT TASK (PUT IN PERMANENT WAIT)
        MOV R6,#HIGH(NOT_TIMING)

IF (PERIODIC_CMD = 1)
        MOV DPTR,#INTVALCNT             ; clear INTERVAL COUNT if task was PERIODIC
        MOV A,RUNNING
        RL A ; DOUBLE THE NUMBER
```

```
        DPTRPLUSA
        MOV A,#0                        ; SAVE 0 in LOW BYTE
        MOVX @DPTR,A
        INC DPTR
        MOVX @DPTR,A                    ; SAVE 0 in HIGH BYTE

        MOV DPTR,#INTVALRLD
        MOV A,RUNNING
        RL A                            ; DOUBLE THE NUMBER
        DPTRPLUSA
        MOV A,#0                        ; SAVE 0 in LOW BYTE
        MOVX @DPTR,A
        INC DPTR
        MOVX @DPTR,A                    ; SAVE 0 in HIGH BYTE
ENDIF

        SJMP SET_TIMEOUT_0

_RESUME:                                ; SYS CALL ENTRY (ONE TASK PARAMETER IN R7)
        CLR EA
        Push_Bank0_Reg
IF (PERIODIC_CMD == 1)
; FIRST CHECK IF THE TASK TO BE RESUMED HAPPENS TO BE A PERIODIC ONE
        MOV DPTR,#INTVALRLD             ; clear INTERVAL COUNT if task was PERIODIC
        MOV A,07
        RL A                    ; DOUBLE THE NUMBER
        DPTRPLUSA
        MOVX A,@DPTR                    ; GET LOW BYTE
```

```
        JNZ NOW_CAN_RESUME
        ; NOW_CHECK HIGH BYTE
        INC DPTR
        MOVX A, @DPTR          ; GET HIGH BYTE
        JZ NOT_A_PERIODIC
; TO RESUME TASK, LOAD INTVALCNT WITH 1 TICK TIME
; SINCE THIS TASK HAD BEEN KILLED, THE INTVALCNT MUST BE ZERO
; AT THIS POINT
NOW_CAN_RESUME:
        MOV DPTR,#INTVALCNT           ; clear INTERVAL COUNT if task was PERIODIC
        MOV A,07
        RL A                   ; DOUBLE THE NUMBER
        DPTRPLUSA
        INC DPTR                      ; GET HIGH BYTE
        MOV A,#1
        MOVX @DPTR,A
        LJMP QSHFT
ENDIF

NOT_A_PERIODIC:
        MOV A, #TTS
        ADD A, RUNNING
        ADD A, RUNNING
        MOV R0,A
        MOV @R0,#1                    ; SET WAITING TIME OF 1 TICK FOR DEFERRED TASK
        INC R0
        MOV @R0,#0                    ; AND THEN SHIFT Q BELOW
        LJMP QSHFT

DEFER:                                ; SYS CALL ENTRY (NO PARAMETERS)
        CLR EA
        Push_Bank0_Reg
        MOV A, #TTS
        ADD A, RUNNING
        ADD A, RUNNING
        MOV R0,A
        MOV @R0,#1             ; SET WAITING TIME OF 1 TICK FOR DEFERRED TASK
        INC R0
        MOV @R0,#0                    ; AND THEN SHIFT Q BELOW

        QSHFT:                 ; SAVE PRESENT RUNNING TASK STACK PTR
        CLR TINQFLAG                  ; CLR TINQFLAG AND SHIFT READYQ BY ONE,
                                      ; GET NEW RUNNING TASK FROM READYQ
        SetBank 1                     ; USE BANK 1 - MAY HAVE ENTERED FROM INTERRUPT
        MOV A, #SPTS                  ; SAVE SP
        ADD A, RUNNING
        MOV R0,A
        MOV @R0, SP                   ; STORE PRESENT STACK POINTER OF TASK
        MOV A,RUNNING
        CALL FETCH_STACK
        Int2Ext                       ; SAVE STACK IN EXTERNAL, READY FOR SWAP
        MOV R1,#(READYQ + 1)          ; NOW SHIFT Q DOWN 1
SHIFT_DOWN:
        MOV A,@R1
        DEC R1
```

```
        MOV @R1,A
        MOV A,R1
        INC R1
        INC R1
        CJNE A,READYQTOP,SHIFT_DOWN
        DEC READYQTOP                ; THEY ALL MOVED DOWN BY 1, HENCE DECREMENT RQTOP
        CJNE A,#READYQ,RUN_NEW_TASK  ; BUT READYQTOP SHOULD NEVER GO BELOW READYQ
        INC READYQTOP                ; SO READYQTOP = READYQ AGAIN, IF IT WAS BELOW
RUN_NEW_TASK:                        ; RUN NEW TASK
        JNB PRIORITY,DONT_SORT       ; DO NOT SORT Q IF PRIORITY OPTION IS OFF
        LCALL TASK_SORT
DONT_SORT:
        MOV A,READYQ
        MOV RUNNING,A                ; SET NEW TASK AS RUNNING
        CALL FETCH_STACK
        Ext2Int                      ; GET NEW STACK IMAGE
        MOV A,#SPTS
        ADD A,RUNNING
        MOV R0,A
        MOV SP,@R0                   ; SET SP TO NEW TASK STACK AREA
        Pop_Bank0_Reg
        SetBank 0
        SETB EA                      ; MAY HAVE ENTERED FROM TIMER INTERRUPT
        RETI                           ; OTHERWISE NO HARM ANYWAY

IF (PERIODIC_CMD = 1)
_PERIODIC:                           ; SYSTEM CALL
        CLR EA
        Push_Bank0_Reg
        MOV DPTR,#INTVALCNT
        MOV A,RUNNING
        RL A                 ; DOUBLE THE NUMBER
        DPTRPLUSA
        MOV A,07                     ; SAVE LOW BYTE, HELD IN R7
        MOVX @DPTR,A
        MOV A,06
        INC DPTR
        MOVX @DPTR,A                 ; SAVE HIGH BYTE, HELD IN R6
        MOV DPTR,#INTVALRLD
        MOV A,RUNNING
        RL A ; DOUBLE THE NUMBER
        DPTRPLUSA
        MOV A,07                     ; SAVE LOW BYTE, HELD IN R7
        MOVX @DPTR,A
        MOV A,06                     ; SAVE HIGH BYTE, HELD IN R6
        INC DPTR
        MOVX @DPTR,A

        Pop_Bank0_Reg
        SETB EA
        RET
ENDIF

TSKRDY_CHK2:                         ; JUST A STEPPING STONE
        LJMP TSKRDY_CHK
```

```
RTOS_TIMER_INT:               ; INTERRUPT ENTRY ONLY FROM TIMER2 OVERFLOW INTERRUPT
                                     ; USES ACC,PSW, (R0,R1 AND R2 FROM BANK 1)
       Push_Bank0_Reg
       SetBank 1                     ; SET TO REGISTERBANK 1
IF (TICK_TIMER = 0)
       CLR TR0                       ; STOP, RELOAD
       MOV TH0,#HIGH(BASIC_TICK)
       MOV TL0,#LOW(BASIC_TICK)
       SETB TR0                      ; AND RESTART TIMER 0
ELSEIF (TICK_TIMER = 1)
       CLR TR1                       ; STOP, RELOAD
       MOV TH1,#HIGH(BASIC_TICK)
       MOV TL1,#LOW(BASIC_TICK)
       SETB TR1                      ; AND RESTART TIMER 1
ENDIF
IF (HALFMSEC = 1)
       JBC MSECFLAG, TSKRDY_CHK2     ; ONLY HALF A MILLISECOND PASSED, HENCE CHK FOR
                                     ; EXTERNAL INTERRUPT TASKS ONLY
       SETB MSECFLAG                 ; USED TO DOUBLE 1/2 MSEC TICKCOUNT DELAY
ENDIF
       DJNZ TICKCOUNT, TSKRDY_CHK2   ; CHECK IF REQUIRED TICK TIME HAS PASSED
       MOV TICKCOUNT, GOPARAM

IF (PERIODIC_CMD = 1)
; FIRST CHECK THE PERIODIC INTERVALS, IF USED
CHK_PERIODICS:
       MOV R0,#0              ; DO ALL TASKS, STARTING WITH TASK 0, HELD IN R0, BANK 1
```

```
        MOV DPTR,#INTVALCNT           ; DPTR POINTS TO FIRST TASK INTERVAL IN TABLE
CHECK_VALS:
        PUSH DPL                      ; SAVE PTR
        PUSH DPH
        LOADREGSXDATA R2,R3           ; R2 = LOW, R3 = HIGH VALUE OF INTVALCNT
        MOV A,R2
        ORL A,R3
        JZ CHECK_NEXTV        ; 0=TASK NOT USING PERIODIC INTERVAL, HENCE SKIP,
                                      ; DO NOT UPDATE INTERVAL COUNT.
COUNT_DOWNV:
        DEC2REGS R2,R3                ; DECREMENT INTERVAL COUNT
        POP DPH                       ; GET POINTER
        POP DPL
        PUSH DPL                      ; AND SAVE POINTER AGAIN
        PUSH DPH
        LOADXDATAREGS R2,R3           ; AND STORE NEW DECREMENTED VALUE
        MOV A,R2
        ORL A,R3
        JNZ CHECK_NEXTV               ; TASK NOT TIMED OUT YET, CHECK NEXT TASK

VAL_OUT:                      ; NEW TASK INTERVAL TIMED OUT, HENCE RELOAD INTERVAL
                                      ; RELOAD VALUE IS NOOFTSKS*2 - 1 AWAY FROM
                                      ; PRESENT DPTR VALUE
        MOV A, #NOOFTSKS
        RL A
        DEC A
        DPTRPLUSA

        LOADREGSXDATA R4,R5
        POP DPH
        POP DPL
        PUSH DPL                      ; SAVE PTR
        PUSH DPH
        LOADXDATAREGS R4,R5

; TEST INTERVAL FLAG
        MOV A,R0
        MOV B,#SIGV                   ; TEST IF SIGNAL ALREADY THERE
        CALL CHK_CLR_FLAG
        JNC SET_VFLAG                 ; NO SIGNAL YET, SO JUST SET FLAG
;
; IF TASK ALREADY IN Q, DO NOT DUPLICATE
; THIS COULD HAPPEN IN CASE OF BAD TASK PROGRAMMING, WHERE
; THE TASK DURATION IS LONGER THAN THE PERIODIC TIME
; IF TASK PROGRAMMING IS OK, THEN THIS CHECK CAN BE ELIMINATED TO REDUCE OVERHEADS.

        MOV R1,#READYQ
CHK_NXT_IN_Q:
        MOV A,@R1
        XRL A,R0
        JZ CHECK_NEXTV
        MOV A,R1
        INC R1
        CJNE A,READYQTOP,CHK_NXT_IN_Q
;
```

```
;
;
        INC READYQTOP                   ; POINT TO TOP OF READY READYQ
        MOV R1,READYQTOP
        MOV @R1,08                      ; PUT TASK# (R0 bank 1 = 08H) IN READYQ
        SETB TINQFLAG                   ; MARK FLAG INDICATING THAT A TASK FINISHED WAITING
IN;TERVAL
        SJMP CHECK_NEXTV                ; AND PLACED IN READYQ
SET_VFLAG:
        MOV A,R0
        MOV B,#SIGV
        CALL SET_FLAG                   ; SET INTERVAL READY BIT
CHECK_NEXTV:
        POP DPH
        POP DPL
        INC DPTR                        ; MOVE UP 1 TASK IN PERIODIC INTERVAL TABLE
        INC DPTR
        INC R0                          ; INCREMENT TASK NUMBER COUNTER
        CJNE R0,#NOOFTSKS,CHECK_VALS    ; END OF ALL TASKS YET?
ENDIF

        SJMP CHK_FOR_TOUTS              ; NOW CHECK FOR TIME OUTS
TSKRDY_CHK1:                            ; JUST A STEPPING STONE
        SJMP TSKRDY_CHK

; NOW CHECK FOR TIME OUTS
CHK_FOR_TOUTS:
        MOV R0,#TTS                     ; R0 POINTS TO FIRST TASK TIMEOUT IN TTS TABLE
        MOV R2,#0                       ; CHECK ALL TASKS, STARTING WITH TASK 0
CHECK_TIMEOUTS:
        MOV A,@R0                       ; GET TIME FOR TASK
        MOV DPL,A
        MOV R1,08                       ; SAVE POINTER TO LOW BYTE IN R1
        INC R0                          ; SAVE POINTER TO HIGH BYTE IN R0
        MOV A,@R0
        MOV DPH,A
        ORL A,DPL
        JZ CHECK_NEXT           ; 0=TASK NOT TIMING, HENCE SKIP, DO NOT DECREMENT TIMEOUT
COUNT_UP:                               ; DPTR NOW CONTAINS TIMEOUT VALUE
        INC DPTR                        ; NOW WE CAN INCREMENT IT
        MOV @R0,DPH                     ; SAVE NEW TIME OUT VALUE, EVEN IF ZERO
        MOV @R1,DPL
        MOV A,DPH                       ; AND CHECK IF TIMED UP (ROLL OVER TO ZERO)
        ORL A,DPL                       ; ACCUMULATOR EQUALS ZERO IF TIMED OUT
        JNZ CHECK_NEXT                  ; TASK NOT TIMED OUT YET, CHECK NEXT TASK
TIMED_OUT:                              ; NEW TASK TIMED OUT, HENCE PLACE IN READYQ
        INC READYQTOP                   ; POINT TO TOP OF READY READYQ
        MOV R1,READYQTOP
        MOV @R1,0AH                     ; PUT TASK# (R2 bank 1 = 0AH) IN READYQ
        SETB TINQFLAG                   ; MARK FLAG INDICATING THAT A TASK FINISHED WAITING
        MOV A,R2
        MOV B,#SIGW
        CALL CLR_FLAG                   ; CLEAR SIGNAL WAITING BIT (IF SET)
CHECK_NEXT:
```

```
        INC R0                          ; MOVE UP 1 IN TTS TABLE
        INC R2                          ; INCREMENT TASK NUMBER COUNTER
        CJNE R2,#NOOFTSKS,CHECK_TIMEOUTS      ; END OF ALL TASKS YET?
TSKRDY_CHK:
        JNB TINQFLAG, EXIT              ; NO TASK ADDED, HENCE EXIT
                                        ; NOTE THAT TINQFLAG CAN BE SET BY 4 ROUTINES,
                                        ; CREATE, SIGNAL, RTOS_TIMER_INT AND XTRA_INT
                                        ; IF CLR (FLAG = 0) => NO NEW TASK PUT IN READYQ
                                        ; IF SET (FLAG = 1) => A NEW TASK HAS BEEN PLACED IN
READYQ.
CAN_CHANGE:
        MOV A,RUNNING                   ; CHECK CURRENT TASK
        CJNE A,#IDLE_TASK,EXIT          ; NOT IN IDLE STATE, SO DO NOT INTERRUPT YET
                                        ; BUT LEAVE TINQFLAG SET FOR THE NEXT RTOS INT. CHECK
                                        ; WHEN THE IDLE_TASK MIGHT BE RUNNING.
        LJMP QSHFT                      ; IDLE AND NEW TASK TIMED OUT, HENCE CHANGE TASK

EXIT:
        Pop_Bank0_Reg
        SetBank 0
        SETB EA
        RETI

XTRA_INT:
IF (USING_INT = 1)
                                        ; EXTRA INTERRUPT SERVICE ROUTINE
                                        ; USED DURING EXTERNAL, TIMER AND SERIAL INTERRUPTS
                                        ; CHECKS BITS SET BY WAITI CALL AND PUTS TASK
```

```
                                    ; WAITING FOR ONE RTOS INTERRUPT IF IT
       Push_Bank0_Reg               ; WAS WAITING FOR THIS EXTERNAL INTERRUPT
                                    ; USES ACC, B, PSW,( R0, R2 AND R3 BANK 1 )
       SetBank 1
       MOV B,XINTMASK               ; GET EXTERNAL INTERRUPT MASK
       CLR A                        ; NOW CHECK IF ANY TASKS WERE WAITING
       CLR INTFLAG                  ; FOR THIS INTERRUPT, STARTING WITH TASK 0
TRY_NXT:
       MOV R2,A                     ; STORE TASK NUMBER IN R2 BANK 1
       CALL CHK_CLR_FLAG
       JNC NOT_YET
       SETB INTFLAG                 ; SET MARKER SHOWING THAT AT LEAST ONE
       MOV A,#TTS                   ; TASK WAS WAITING FOR THIS INTERRUPT
       ADD A,R2
       ADD A,R2
       MOV R0,A                     ; HENCE
       MOV @R0,#LOW(NOT_TIMING)     ; MARK TASK AS NOT WAITING
       INC R0
       MOV @R0,#HIGH(NOT_TIMING)    ; MARK TASK AS NOT WAITING
       INC READYQTOP                ; AND
       MOV R0,READYQTOP             ; PUT FOUND TASK ON READYQ
       MOV @R0,0AH                  ; QSHFT WILL DO THE REST LATER.
NOT_YET:
       MOV A,R2
       INC A                        ; CHECK NEXT TASK
       CJNE A,#NOOFTSKS,TRY_NXT
       JNB INTFLAG, EXIT_INT        ; NO TASK FOUND WAITING INT, HENCE EXIT
EXIT_INT_SHFT:
       SETB TINQFLAG ; INDICATE THAT A NEW TASK HAS BEEN PUT IN READY Q
       MOV A,RUNNING                ; CHECK CURRENT TASK
       CJNE A,#IDLE_TASK,EXIT_INT   ; NOT IN IDLE STATE, SO DO NOT SHIFT TASKS
                                    ; BUT TINQFLAG WILL STILL REMAIN SET SO THAT THE
                                    ; RTOS_TIMER_INT ROUTINE CAN HANDLE IT LATER.
OK2SHFT:
       LJMP QSHFT
EXIT_INT:
       Pop_Bank0_Reg
       SetBank 0
       SETB EA
ENDIF

       RETI
; SUB ROUTINES USED IN THE RTOS

; ***********************************************
SET_FLAG:
; ENTRY A = TASK NUMBER
;       B = BIT MASK
; EXIT REQUIRED BIT IN TASK FLAG BYTE SET
  PUSH 00
  PUSH 08
  MOV R0,#TSKFLAGS
  ADD A,R0
  MOV R0,A
```

```
  MOV A,@R0
  ORL A,B               ; SET REQUIRED BIT TO 1
  MOV @R0,A
  POP 08
  POP 00
  RET

; *********************************************
CHK_FLAG:
; ENTRY A = TASK NUMBER
;       B = BIT MASK
; EXIT CARRY SET IF FLAG WAS FOUND TO BE SET
  PUSH 00
  PUSH 08
  MOV R0,#TSKFLAGS
  ADD A,R0
  MOV R0,A
  MOV A,@R0
  CLR C
  ANL A,B
  JZ EXIT1             ; BIT WAS CLEARED, CARRY = 0
  SETB C               ; BIT WAS SET, CARRY =1
EXIT1:
  POP 08
  POP 00
  RET

; *********************************************
CLR_FLAG:
CHK_CLR_FLAG:
; BOTH NAMES CORRESPOND TO THE SAME ROUTINE
; ENTRY A = TASK NUMBER
;       B = BIT MASK
; EXIT CARRY SET IF FLAG WAS FOUND TO BE SET
;       AND THEN CLEARS FLAG BEFORE EXITING ROUTINE
;       CARRY BIT = 0 IF BIT WAS ZERO
  PUSH 00
  PUSH 08
  MOV R0,#TSKFLAGS
  ADD A,R0
  MOV R0,A
  MOV A,@R0
  CLR C
  ANL A,B
  JZ EXIT2             ; BIT WAS CLEAR, HENCE EXIT, CARRY = 0
  MOV A,@R0
  XRL A,B              ; SINCE IT WAS SET, THEN SIMPLY XOR WITH MASK
  MOV @R0,A            ; TO CLEAR AND STORE
  SETB C               ; CARRY = 1 SINCE BIT WAS INITIALLY SET
EXIT2:
  POP 08
  POP 00
  RET
```

```
; **********************************************
FETCH_STACK:
; ENTRY A = TASK NUMBER, USES ACC, DPTR, AND R0
; EXIT DPTR POINTS TO START OF STACK AREA FOR TASK
  MOV TMPSTORE0,A
  MOV DPTR,#EXT_STK_AREA
  MOV R0,#0
LOOP1:
  MOV A,R0
  CJNE A,TMPSTORE0,CONT1
  RET
CONT1:
  MOV A,#STACKSIZE
  ADD A,DPL
  MOV DPL,A
  MOV A,DPH
  ADDC A,#0
  MOV DPH,A
  INC R0
  SJMP LOOP1

; **********************************************
; SORT THE READY Q, LOW TASK NUMBER IS THE HIGHEST
; PRIORITY, AND THEREFORE AFTER ONE Q SORT PASS,
; THE LOWEST NUMBERED TASK ENDS UP AT BOTTOM OF Q,
; NEXT IN LINE TO EXECUTE.
; IT IS CALLED FROM QSHFT, WHEN REGISTER BANK 1 IS BEING USED.
;
TASK_SORT:
  PUSH ACC
  PUSH 08
  PUSH B
  MOV R0,READYQTOP             ; R0 POINTS IN READY Q AREA
  MOV A,R0
  CJNE A,#READYQ,NEXT_PAIR
  SJMP EXIT_QSORT              ; ONLY ONE TASK, HENCE EXIT
NEXT_PAIR:
  MOV A,@R0
  MOV B,@R0
  DEC R0
  CLR C
  SUBB A,@R0
  JNC NO_SWAP                  ; ENSURE LOWEST TASK NUMBER (HIGHEST PRIORITY)
                               ; TRICKLES DOWN TO READYQ BOTTOM, READY TO RUN
SWAP_NOW:
  MOV A,@R0
  MOV @R0,B
  INC R0
  MOV @R0,A
  DEC R0
NO_SWAP:
  CJNE R0,#READYQ,NEXT_PAIR   ; ONE PASS DONE, HENCE EXIT
EXIT_QSORT:
  POP B
```

```
  POP 08
  POP ACC
  RET
; *********************************************
END
```

Other Packages

SerIntPrPkg.c

```
/* SerIntPrPkg.c – see remarks below for program dedtails   */

/* Has a 200 byte Receive and a 200-byte Transmit buffer in XDATA   */
/* Routines to use with C program when using the on-board UART      */
/* Running under interrupt control, using a stand-alone ISR, not under RTOS */
/* auto baud rate detection used by using a timer to count the bit time */

/* If Baudrate supplied is 0, then Auto Baud Detection is performed */

#include <reg52.h>            /* special function registers 8052 */
#include <absacc.h>
#include <stdio.h>

// RXD is bit 0xB0;           /* Rx Data on internal UART is Port 3 bit 0 */

#define RX_BUFFER_LENGTH 200
#define TX_BUFFER_LENGTH 200

unsigned char xdata Rx_buffer[RX_BUFFER_LENGTH]; /* software Receive buffer */
unsigned char xdata Tx_buffer[TX_BUFFER_LENGTH]; /* software Transmit buffer */
unsigned char data In_read_index;
/* points to data in software buffer that has been read */
unsigned char data In_waiting_index;
/* points to data in software buffer not yet read */
unsigned char data Out_written_index;
/* points to data in software buffer that has been sent */
unsigned char data Out_waiting_index;
/* points to data in software buffer not yet sent */


void Init_P3_Int (unsigned int baudrate);
unsigned int autobaud(void);
void uart_P3_isr (void);
/* This should be created as a function, waiting for serial interrupt */
char putchar (char c);
char _getkey (void);   /* This preferably should not be a Wait for Key routine */
                       /* It must have some TimeOut facility not to hold other jobs */
/* ========================================== */
void Init_P3_Int (unsigned int baudrate){
unsigned int autobaud(void);

ET1 = 0;               /* Disable Timer 1 interrupt just in case */
ES = 0;                /* Disable Serial Interrupt initially just in case. */
                       /* It will then be enabled by the main program */

if (baudrate==0) baudrate = autobaud();

SCON = 0x50;                  /* Setup serial port control register */
                              /* Mode 1: 8-bit uart var. baud rate */
                              /* REN: enable receiver, TI=0 */

PCON &= 0x7F;                 /* Clear SMOD bit in power ctrl reg (no double baudrate) */

TMOD &= 0x0F;                 /* Setup timer/counter mode register */
                              /* Clear M1 and M0 for timer 1 */
```

```
TMOD |= 0x20;                  /* Set M1 for 8-bit auto-reload timer mode 2 */

RCLK = 0;                      /* USE TIMER 1 FOR RECEIVE BAUD RATE (8032 only) */
TCLK = 0;                      /* USE TIMER 1 FOR TRANSMIT BAUD RATE (8032 only) */

switch (baudrate) {
  case 300:
        TH1 = TL1 = 0xA0;
        break;
  case 600:
        TH1 = TL1 = 0xD0;
        break;
  case 1200:
        TH1 = TL1 = 0xE8;
        break;
  case 2400:
        TH1 = TL1 = 0xF4;
        break;
  case 4800:
        TH1 = TL1 = 0xFA;
        break;
  case 9600:
        TH1 = TL1 = 0xFD;
        break;
  case 19200:
        TH1 = TL1 = 0xFD;
        PCON |= 0x80;          /* double baudrate, SMOD = 1 */
        break;
  case 57600:
        TH1 = TL1 = 0xFF;      /* Not quite standard */
        PCON |= 0x80;          /* double baudrate, SMOD = 1 */
        break;
}

In_read_index = In_waiting_index = 0;        /* Reset Receive buffer pointers */
Out_written_index = Out_waiting_index = 0;   /* Reset Transmit buffer pointers */

TR1 = 1;        /* Start timer 1 for baud rate generation */
ES = 1;         /* Enable serial interrupt */
TI = RI = 0;    /* Clear TI and RI */
EA = 1;         /* Enable global interrupts */
}

/* Autobaud Calculation */
/* Calculates the time for 2 bits (the Start bit and the least significant bit, */
/* which should be a 1 */
/* Assuming you press the ENTER key (13 decimal = 00001101 binary) */
/*                                                                           */
/*             0 1 0 1 1 0 0 0 0 1                                           */
/*             | |             | |                                           */
/* start bit--->+ +<--lsb msb-->+ +<---stop bit                    */

unsigned int autobaud(void){

unsigned char data i;
unsigned int data counter;
```

```
unsigned int code count_table[] = {16,64,144,288,576,1152,2304,4608,9216,65535};
unsigned int code baud_table[] = {0,57600,19200,9600,4800,2400,1200,600,300,0};
// Counter running at a rate of 1 count every 12/11.0592 micro seconds
// Count reached after a time of 2 bits (bitrate =br) is (2*11059200)/(12*br)
//              = 1843200/br
//                                Upper Limit       = 65535  (invalid baudrate)
//                          300   Upper boundary    = 9216  <=======
// if 300 baud, count reached after 2 bits would be 6144
//                          600   boundary          = 4608  <=======
// if 600 baud, count reached after 2 bits would be 3072
//                          1200  Upper boundary    = 2304  <=======
// if 1200 baud, count reached after 2 bits would be 1536
//                          2400  boundary          = 1152  <=======
// if 2400 baud, count reached after 2 bits would be 768
//                          4800  boundary          =  576  <=======
// if 4800 baud, count reached after 2 bits would be 384
//                          9600  boundary          =  288  <=======
// if 9600 baud, count reached after 2 bits would be 192
//                          19200 boundary          =  144  <=======
// if 19200 baud, count reached after 2 bits would be 96
//                          57600 boundary          =   64  <=======
// if 57600 baud, count reached after 2 bits would be 32
//                                Lower boundary    =   16  (invalid baudrate)

do {
  TMOD &= 0x0F;        /* Setup timer/counter mode register */
                       /* Clear M1 and M0 for timer 1 */
  TMOD |= 0x10;        /* Set M0 for 16-bit timer mode 1 */
```

```
  TH1 = TL1 = 0;       /* Load counter registers with zero */
  while(RXD){};        /* wait for start bit (loop until RXD = 0) */
  TR1 = 1;             /* Start timing */
  while(!RXD){};       /* wait for start bit to finish */
  while(RXD){};        /* wait for 1st one bit (RXD = 1) */
  TR1 = 0;             /* Stop count => has value of 2 bits */
  counter = TH1*256 + TL1;    /* Calculate the count value */
  i=0;
  while (counter > count_table[i]) {
               i++;           // Find entry in table to correspond with this count
                       }              // Valid i values are 1 to 8 only.
  } while (i==0 || i==9);     // Upper and Lower values of table are not valid
                              // hence loop until found valid baud rate
return(baud_table[i]);
}

/* The following is the UART Interrupt Service Routine */
/* It should be run as a function under serial interrupt */
/*******************************************************************************/
/*      'UART P3 ISR':                                                         */
/*******************************************************************************/
void uart_isr (void) interrupt 4 using 1
{       /* Runs as a Serial Interrupt Routine */
  /* Wait for Serial Interrupt number 4 */
  /* Check if interrupt from TI or RI */
  /* TI set by 8032 UART whenever a character has just been transmitted */
  /* RI is set by the 8032 whenever a complete character has been received in SBUF */
  /* BOTH RI and TI should be reset by the software */

 if (TI)
       {                              /* Check if interrupt from TI */
/* Transmitter section */
       TI=0;                          /* Transmitter is ready, hence */
                                      /* prepare for next transmission */
                                      /* Check if there is anything else to transmit */
       if (Out_written_index < Out_waiting_index)
          SBUF = Tx_buffer[Out_written_index++];
               /* put data in hardware buffer for Tx */
       else
         {                    /* No new data to send, just reset Tx buffer index */
         Out_waiting_index = 0;
         Out_written_index = 0;
         }
 }

 if (RI)
       {                              /* Check if interrupt from RI */
/* Receiver Section = Flag set when full character has been received */
       RI=0;
/* If all old data in software buffer has been read, */
/* we can start reading again into index 0 and reset RX buffer index */
```

```
        if (In_waiting_index == In_read_index)
                {
                In_waiting_index = 0;
                In_read_index = 0;
                }
/* Read the data from the UART hardware buffer (SBUF) into the software buffer */
        Rx_buffer[In_waiting_index] = SBUF;
        if (In_waiting_index < RX_BUFFER_LENGTH)
                In_waiting_index++;
        }
}

char putchar (char c)
{
// Writes to software buffer ONLY if there is space.
// If no space, keep on trying for a short TimeOut period
// No error reporting in this simple library
unsigned int data TimeOut = 20000;
        while ((Out_waiting_index >= TX_BUFFER_LENGTH) && (TimeOut-- > 0));
 // wait for buffer space only for a TimeOut period
          if (Out_waiting_index < TX_BUFFER_LENGTH)
                {
                Tx_buffer[Out_waiting_index++] = c;
          TI = 1;      /* Generate interrupt - Activate TI to start transmission */
                }
          return (c);
}

char _getkey ()
{
// No permanent waiting for key press - just waits for a time-out period
// Retrieves a character from the software buffer, if available.
// The character from the buffer is returned, or if no character
// is available, a 0 is returned.
        unsigned char data c = 0;     // Zero is returned if no key is pressed
        unsigned int data TimeOut = 20000;

        while ((In_read_index >= In_waiting_index) && (TimeOut-- > 0));
        // if no new character available, wait for a short TimeOut period
                if (In_read_index < In_waiting_index)
                {
                c = Rx_buffer[In_read_index];
                if (In_read_index < RX_BUFFER_LENGTH)
                        In_read_index++;
                }
        return (c);           // Returns NULL if no new character received
}
/******************************************************************************/
```

SerP2Pkg.c

```
/**************************************************************************/
/* SerP2Pkg.c */
/* Routines to use with C program when using the additional SCC2691 UART (P2) */
/* on the FLT-32 board. */

/* If baud rate parameter given is zero, auto-baudrate detection is performed */

#include <reg52.h>                    /* special function registers 8052      */
#include <absacc.h>
#include <stdio.h>

void P2_SetUp(void);
unsigned int Auto_P2_BaudRate(void);
void Set_P2_BaudRate(unsigned int baud);

char putchar (char c);
char _getkey (void);

#define RX      XBYTE [0xFFE8]        // READ RX DATA AT START-UP (RxD Pin on P2)
                                      //      (READ BIT 0 OF THIS ADDRESS)
                                      //      USED FOR AUTO-BAUD DETECTION ONLY
                      //                    UART BASE ADDRESS on Flite-32 is FFF8H
#define UART_MR1      XBYTE [0XFFF8] // MR1 - Mode Register 1
#define UART_MR2      XBYTE [0XFFF8] // MR2 - Mode Register 2
#define UART_SR       XBYTE [0XFFF9] // READ SR - Channel Status Register
#define UART_CSR      XBYTE [0XFFF9] // WRITE CSR - Clock Select Register
#define UART_CR       XBYTE [0XFFFA] // WRITE CR - Command Register
```

```
#define UART_RHR        XBYTE [0XFFFB] // READ RHR - Receive Holding Register
#define UART_THR        XBYTE [0XFFFB] // WRITE THR - Transmit Holding Register
#define UART_ACR        XBYTE [0XFFFC] // WRITE ACR - Auxiliary Control Register
#define UART_ISR        XBYTE [0XFFFD] // READ ISR - Interrupt Status Register
#define UART_IMR        XBYTE [0XFFFD] // WRITE IMR - Interrupt Mask Register
#define UART_CTU        XBYTE [0XFFFE] // READ/WRITE CTU - Counter Timer Upper Register
#define UART_CTL        XBYTE [0XFFFF] // READ/WRITE CTL - Counter Timer Lower Register

/*************************************************************************************/

void P2_SetUp(){
unsigned char c;

do {
  UART_CR      = 0x2A;        // reset Rx. Rx and Tx disabled
  UART_CR      = 0x3A;        // reset Tx. Rx and Tx disabled
  UART_CR      = 0x4A;        // reset Error Status. Rx and Tx disabled
  UART_CR      = 0x1A;        // reset MR pointer. Rx and Tx disabled
  UART_MR1     = 0x13;        // 8 bit, no parity
  UART_MR2     = 0x07;        // 1 stop bit
  UART_ACR     = 0x38;        // BRG=0. Set 1 of baud rate table.
                                             // Counter i/p xtal/16
  UART_CSR     = 0xCC;        // 38400 baud
  UART_IMR     = 0x00;        // No interrupts
  UART_CR      = 0x16;        // reset MR pointer. Rx and Tx enabled
  } while (((c=UART_SR) & 0x04) == 0); // Repeat setup if TX not yet ready
  UART_CR      = 0x2A;        // reset Rx. Rx and Tx disabled
  UART_CR      = 0x3A;        // reset Tx. Rx and Tx disabled
  UART_CR      = 0x4A;        // reset Error Status. Rx and Tx disabled
}

unsigned int Auto_P2_BaudRate(void){ // detect and return baud rate
unsigned char c,d,i;
unsigned int counter;
unsigned int code count_table[] = {9,18,36,72,144,288,576,1152,2304,3630,5200,65535};
unsigned int code baud_table[] = {0,38400,19200,9600,4800,2400,1200,
                              600,300,150,110,0};

// Counter running at a rate of 1 count every 16/3.6864 micro seconds
// Count reached after a time of 2 bits (bitrate=br) is (2*3686400)/(16*br)
//                          Invalid Maximum limit     = 65535 <=========
//                              110    boundary       = 5200 <=========
// if 110 baud, count reached after 1 bit would be 4189
//                              150     boundary       = 3630 <=========
// if 150 baud, count reached after 1 bit would be 3072
//                              300     boundary       = 2304 <=========
// if 300 baud, count reached after 1 bit would be 1536
//                              600     boundary       = 1152 <=========
// if 600 baud, count reached after 1 bit would be 768
//                              1200    boundary       = 576 <=========
// if 1200 baud, count reached after 1 bit would be 384
//                              2400    boundary       = 288 <=========
// if 2400 baud, count reached after 1 bit would be 192
//                              4800    boundary       = 144 <=========
// if 4800 baud, count reached after 1 bit would be 96
//                              9600    boundary       = 72 <=========
```

```
// if 9600 baud, count reached after 1 bit would be 48
//                            19200   boundary       = 36 <=========
// if 19200 baud, count reached after 1 bit would be 24
//                            38400   boundary       = 18 <=========
// if 38400 baud, count reached after 1 bit would be 12
//                        Invalid lower boundary     = 9 <=========

do {
  P2_SetUp();
  UART_CTU     = 0xFF;                              // Reset counter to 65535
  UART_CTL     = 0xFF;
  while (((c=RX) & 0x01) != 0){};     // wait for start bit
  UART_CR      = 0x8A;                              // start counter
// counting duration of 2 bits - the start buit and another '1' bit
  while (((c=RX) & 0x01) == 0){};     // wait for start bit to pass
  while (((c=RX) & 0x01) == 1){};     // wait for '1' bit to pass
  UART_CR      = 0x9A;                              // stop counter (now holding count for 2
bits)
  c = ~(c = UART_CTU);                       // Since counter counts down, we have to
  d = ~(d = UART_CTL);                       // complement the readings.
  counter = 256*c + d;                       // Get counter value
  i=0;
  while (counter > count_table[i]) {
              i++;           // Find entry in table to correspond with this count
                                    } // Valid i values are between 1 and 10 ONLY.
  } while (i==0 | i==11);             // upper and lower values of table are not valid

  return(baud_table[i]);
}

void Set_P2_BaudRate(unsigned int baud) {
unsigned char c;

  if (baud==0) baud = Auto_P2_BaudRate();

  UART_CR      = 0x2A;               // reset Rx. Rx and Tx disabled
  UART_CR      = 0x3A;               // reset Tx. Rx and Tx disabled
  UART_CR      = 0x4A;               // reset Error Status. Rx and Tx disabled
  UART_CR      = 0x1A;               // reset MR pointer. Rx and Tx disabled
  UART_MR1     = 0x13;               // 8 bit, no parity
  UART_MR2     = 0x07;               // 1 stop bit
  UART_ACR     = 0xB8;               // BRG=1. Set 2 of baud rate table.

  switch(baud) {
       case 110:
              UART_CSR = 0x11;
                     break;
       case 150:
              UART_CSR = 0x33;
                     break;
       case 300:
              UART_CSR = 0x44;
                     break;
```

```
        case 600:
                UART_CSR = 0x55;
                         break;
        case 1200:
                UART_CSR = 0x66;
                         break;
         case 2400:
                 UART_CSR = 0x88;
                         break;
        case 4800:
                UART_CSR = 0x99;
                         break;
        case 9600: default:
                UART_CSR = 0xBB;
                         break;
         case 19200:
                UART_CSR = 0xCC;
                         break;
        case 38400:
                UART_ACR = 0x38;
                UART_CSR = 0xCC;
                         break;
        }
 UART_CR        = 0x9A;                       // Stop Counter
 UART_CTU       = 0xFF;                       // Reload counter with 65535
 UART_CTL       = 0xFF;
 UART_CR        = 0x1A;                       // reset MR pointer. Rx and Tx disabled
 UART_MR1       = 0x13;                       // 8 bit, no parity
 UART_MR2       = 0x07;                       // 1 stop bit
 UART_CR        = 0x05;                       // Enable Tx and Rx
 UART_CR        = 0x85;                       // Start Counter
 while (((c=UART_ISR) & 0x10) == 0){};
               // Wait for counter to reach zero (just a delay)
 UART_CR        = 0x95;                       // Stop Counter
 while (((c=UART_SR) & 0x01) == 1){
 c=UART_RHR;                                  // clear receive FIFO buffer
        }
 UART_CR        = 0x45;                       // Reset Error Status. Rx and Tx enabled
}

char putchar (char c) {
unsigned char d;
        while (((d=UART_SR) & 0x04) == 0){};
        UART_THR = c;
        return (c);
        }

#if 1
char _getkey (void){ // wait for key for ever
char c;
        while (((c=UART_SR) & 0x01) == 0){};
        return(c=UART_RHR);
}
```

```
#endif

#if 0
char _getkey (void){   // wait for key with TIMEOUT
char c;
unsigned long TimeOutLoop = 12000;    // just as a test
       while ((((c=UART_SR) & 0x01) == 0) && (++TimeOutLoop !=0)){};
                       // wait for character or TimeOut
       if (((c=UART_SR) & 0x01) == 1)
               return(c=UART_RHR);
       else
               return (0);
}
#endif

#if 0
char _getkey (void){ // no waiting
char c;
       if (((c=UART_SR) & 0x01) == 1)
               return(c=UART_RHR);
       else
               return (0);
}
#endif
/* =================================================== */
```

SerP3Pkg.c

```
/* ==================================================== */
/* SerP3Pkg.c - see remarks below for program details */
/* Routines to use with C program when using the 8032 on-board UART (P3)*/
/* NOT under interrupt control */
/* Uses Timer 1 for Baud rate Generation */
/* Serial and Timer 1 interrupts are disabled. */

/* If Baudrate supplied is 0, then Auto Baud Detection is performed */

#include <reg52.h>            /* special function registers 8052 */
#include <absacc.h>
#include <stdio.h>

// RXD is bit 0xB0;    /* Rx Data on internal UART is Port 3 bit 0 */

void Set_P3_BaudRate (unsigned int baudrate);
unsigned int P3autobaud(void);
char putchar (char c);
char _getkey (void);

/* ========================================== */
void Set_P3_BaudRate (unsigned int baudrate){
/* NOT under interrupt control */
if (baudrate==0) baudrate = P3autobaud();

SCON = 0x50;   /* Setup serial port control register */
               /* Mode 1: 8-bit uart var. baud rate */
               /* REN: enable receiver */

PCON &= 0x7F;  /* Clear SMOD bit in power ctrl reg */
TMOD &= 0x0F;  /* Setup timer/counter mode register */
               /* Clear M1 and M0 for timer 1 */
TMOD |= 0x20;  /* Set M1 for 8-bit auto-reload timer 1 */

RCLK = 0;      /* USE TIMER 1 FOR RECEIVE BAUD RATE */
TCLK = 0;      /* USE TIMER 1 FOR TRANSMIT BAUD RATE */
switch (baudrate) {
       case 300:
              TH1 = TL1 = 0xA0;
              break;
       case 600:
              TH1 = TL1 = 0xD0;
              break;
       case 1200:
              TH1 = TL1 = 0xE8;
              break;
       case 2400:
              TH1 = TL1 = 0xF4;
              break;
       case 4800:
              TH1 = TL1 = 0xFA;
              break;
```

```
        case 9600:
                TH1 = TL1 = 0xFD;
                break;
        case 19200:
                TH1 = TL1 = 0xFD;
                PCON |= 0x80;          /* double baudrate, SMOD = 1 */
                break;
        case 57600:
                TH1 = TL1 = 0xFF;      /* Not quite standard */
                PCON |= 0x80;          /* double baudrate, SMOD = 1 */
                break;
        }

ET1 = 0;                /* Disable timer 1 interrupts just in case */
ES = 0;                 /* Disable serial interrupts just in case */
TR1 = 1;                /* Start timer 1 */
TI = 1;                 /* Set TI to indicate ready to xmit */
}

/* Autobaud Calculation */
/* Calculates the time for 1 bit (Start bit only) */
/* Assuming you press the ENTER key (13 decimal = 00001101 binary) */
/*
                */
/*              0 1 0 1 1 0 0 0 0 1                     */
/*              | | | |                                 */
/* start bit--->+ +<--lsb msb-->+ +<---stop bit                 */
/* For the baud rate to be detected properly, the key pressed */
/* should have its LSB = 1 (ALL ODD ASCII characters such as A, a) */

unsigned int P3autobaud(void){

unsigned char i;
unsigned int counter;
unsigned int code count_table[] = {10,32,72,144,288,576,1152,2304,3200,65535};
unsigned int code baud_table[] = {0,57600,19200,9600,4800,2400,1200,600,300,0};
// Counter running at a rate of 1 count every 12/11.0592 micro seconds
// Count reached after a time of 1 bit (bitrate=br) is (1*11059200)/(12*br)
//              = 921600/br
//                      Invalid Maximum limit         = 65535 <=========
//                              300     boundary       = 3200 <=========
// if 300 baud, count reached after 1 bit would be 3072
//                              600     boundary       = 2304 <=========
// if 600 baud, count reached after 1 bit would be 1536
//                              1200    boundary       = 1152 <=========
// if 1200 baud, count reached after 1 bit would be 768
//                              2400    boundary       = 576 <=========
// if 2400 baud, count reached after 1 bit would be 384
//                              4800    boundary       = 288 <=========
// if 4800 baud, count reached after 1 bit would be 192
//                              9600    boundary       = 144 <=========
// if 9600 baud, count reached after 1 bit would be 96
//                              19200   boundary       = 72 <=========
```

```
// if 19200 baud, count reached after 1 bit would be 48
//                            57600   boundary       = 32 <=========
// if 57600 baud, count reached after 1 bit would be 16
//             Invalid lower         boundary       = 10 <=========

do {
 TMOD &= 0x0F;         /* Setup timer/counter mode register */
                       /* Clear M1 and M0 for timer 1 */
 TMOD |= 0x10;         /* Set M0 for 16-bit timer mode 1 */
 TH1 = TL1 = 0;        /* Load counter registers with zero */
 ET1 = 0;              /* Disable timer 1 interrupts just in case */
 ES = 0;                      /* Disable serial interrupts just in case */
 while(RXD){};         /* wait for start bit (loop until RXD = 0) */
 TR1 = 1;              /* Start timing */
 while(!RXD){};        /* wait for start bit to finish */
 TR1 = 0;              /* Stop count => has value of 1 bit */
 counter = TH1*256 + TL1; /* Calculate the count value */
 i=0;
 while (counter > count_table[i]) {
       i++;            // Find entry in table to correspond with this count
                       }                      // Valid i values are 1 to 8 only.
 } while (i==0 || i==9);              // upper and lower values of table are not valid
                                      // hence loop until found valid baud rate
return(baud_table[i]);
}

/*
 * putchar (mini version): outputs charcter only
 */
char putchar (char c) {
 while (!TI);          // If TI=0, previous transmission not yet ready, hence wait
 TI = 0;
 return (SBUF = c);
}

#if 0
char _getkey () {             // no interrupt, no waiting for key press
 char c=0;
       if (RI)
  {
  c = SBUF;
  RI = 0;
  }
 return (c);
}
#endif

#if 0
char _getkey () {      // no interrupt, wait for key press for a time out period
 char c=0;
 unsigned long TimeOutLoop = 12000;          // just as a test

 while ((!RI) && (++TimeOutLoop !=0)){}; // wait for character or TimeOut
```

```
  if (RI)
  {
  c = SBUF;
  RI = 0;
         }
 return (c);
}
#endif

#if 1
char _getkey () {      // no interrupt, wait for key press for ever
 char c;

 while (!RI); // wait here until character received
 c = SBUF;
 RI = 0;
 return (c);
}
#endif

/******************************************************************************/
```

Flt32Pkg.c

```
/* ==================================================== */

/* Flt32Pkg.c */
/* Routines to use with C programs */
/* Used to initialise 8255, read and write from ports */
/* when using the add-on Applications Board */
/* with the FLT-32 board. */

#include <reg52.h>            /* special function registers 8052    */
#include <absacc.h>

void Init_8255(unsigned char c);
void WritePort(unsigned char c, unsigned char d);
unsigned char ReadPort(unsigned char c);

#define PortA           XBYTE [0XFF40]         // 8255 Port A register
#define PortB           XBYTE [0XFF41]         // 8255 Port B register
#define PortC           XBYTE [0XFF42]         // 8255 Port C register
#define Control         XBYTE [0XFF43]         // 8255 Control register

/*************************************************************************************/

void Init_8255 (unsigned char c){
 Control = c;          // set-up 8255 PIO chip
}

/*************************************************************************************/

void WritePort (unsigned char c, unsigned char d)
       {
         switch (c)
         {
                case 'A': case 'a':
                 PortA = d;
                 break;
                case 'B': case 'b': default:
                 PortB = d;
                 break;
                case 'C': case 'c':
                 PortC = d;
                 break;
                }
       }

/*************************************************************************************/

unsigned char ReadPort (unsigned char c)
       {
         unsigned char d;
         switch (c)
         {
                case 'A': case 'a': default:
```

```
                d = PortA;
                break;
               case 'B': case 'b':
                d = PortB;
                break;
               case 'C': case 'c':
                d = PortC;
                break;
               }
       return (d);
       }

/* ==================================================== */
```

RTOS Example Programs

Using the PaulOS RTOS under the Keil environment is similar to using it to write a normal C program. The files required are shown in the screen shot below in figure B-1. Note that RTMACROSV5C.A51 and PaulosV5C.A51 are declared as text files (right click on them and check the options). This is because they are 'included' in TaskStkV5C.A51 file. Note also that the NOFTSKS has to agree with the number of tasks in the application and MAIN_STACK has to agree with the ?STACK value given in the AAA.m51 list file generated by the compiler.

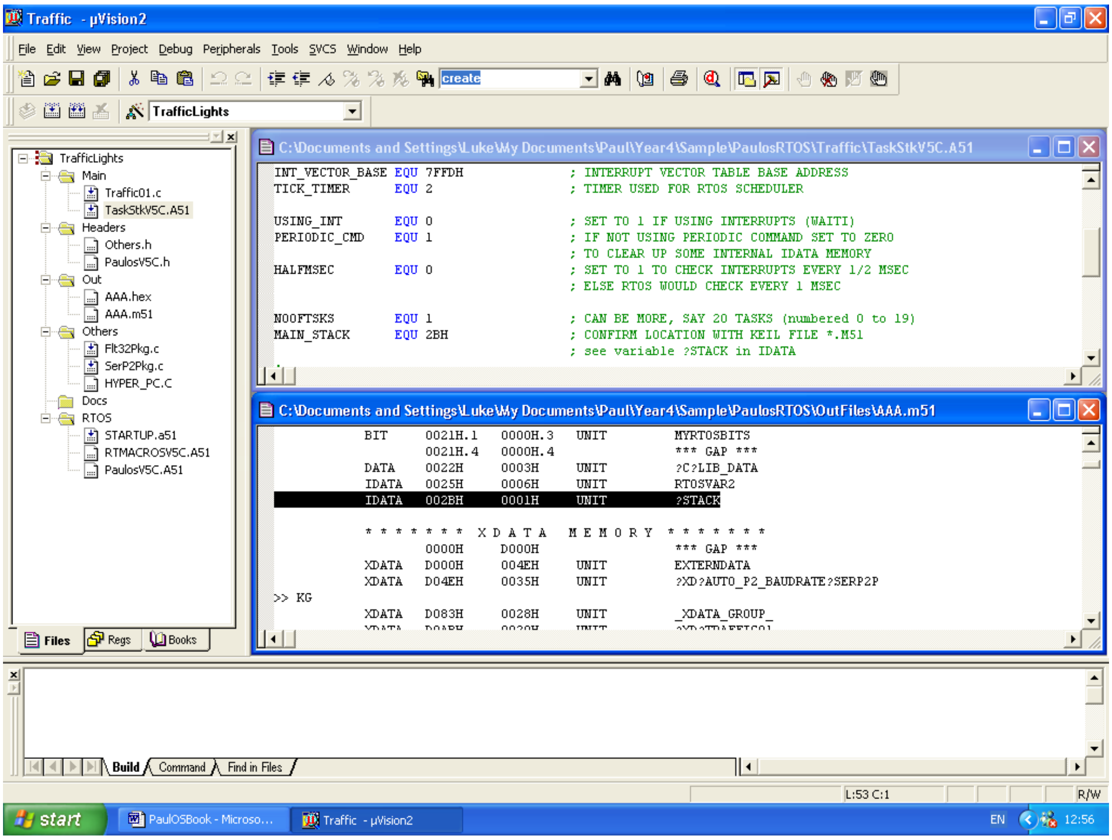


Figure B-1 Keil Screen shot using PaulOS RTOS

The following are some example programs using Paulos rtos so that you can have an idea of its capabilities and syntax.

Light Controller Example Program

```
/*****************************************************************************/
/*                              RTOS */
/* Paul_19.c: MULTI-TASK Light Controller using the C-51 COMPILER            */
/* USE LARGE MODEL, WITH EXTERNAL DATA                                        */
/* Application Program using Paulos.a51                                       */
/* RTOS program                                                               */
/* Make sure to include the Paulos.a51 program                                */
/*      with the source group                                                 */
/*      together with the Paulos.h header file                                */
/*      (defining the system calls)                                           */
/* All tasks run in bank 0, RTOS kernel runs in bank 1                        */
/* All tasks must be written as an endless loop.                              */
/* VALID INTERRUPT NUMBERS ARE AS FOLLOWS:                                    */
/*             0   EXTERNAL INT 0     (IEMASK = 00100001 = 21H)               */
/*             1   TIMER COUNTER 0    (IEMASK = 00100010 = 22H)               */
/*             2   EXTERNAL INT 1     (IEMASK = 00100100 = 24H)               */
/*             3   TIMER COUNTER 1    (IEMASK = 00101000 = 28H)               */
/*             4   SERIAL PORT        (IEMASK = 00110000 = 30H)               */
/*             5   TIMER COUNTER 2    (IEMASK = 00100000 = 20H)               */
/*      TIMER 2 IS USED BY THE RTOS, AND ET2 (20H) SHOULD BE ALWAYS SET       */
/*                    HENCE THE OTHER IEMASKS ARE ALWAYS ORED WITH 20H        */
/* */
/*****************************************************************************/
//
//             N O T E
//   USE the following settings in Options for Target 1
// Memory Model: LARGE: EXTERNAL XDATA
// Code Model: LARGE: 64K Program
//         START       SIZE
// CODE:   0X8100      0X0A00
// RAM:    0X8B00      0X1500

#include <reg52.h>                   /* special function registers 8052 */
#include "Paulos.h"                  /* Paul RTOS system calls definitions */
#include <absacc.h>
#define porta   XBYTE [0xFF40]       /* 8255 port mappings on FLT-32 */
#define portb   XBYTE [0xFF41]
#define portc   XBYTE [0xFF42]
#define control XBYTE [0xFF43]
#define LEDS portb

unsigned char data display=0;        /* place display variable in internal 'data' RAM */

/*****************************************************************************/
/*      Task 0 'lights0':                                                     */
/*****************************************************************************/
void lights0 (void){                 /* LED operation      */
       while(1){
         display=display ^ 0x01;
       WAITT(1);
         SIGNAL(1);           /* SIGNAL TO TASK 1    */
              WAITS(255);            /* WAIT FOR A SIGNAL INDEFINITELY     */
              }
 }
```

```
/*****************************************************************************/
/*****************************************************************************/
/*      Task 1 'lights1': */
/*****************************************************************************/
void lights1 (void){          /* LED operation       */
        while(1){
              WAITS(255);
          display=display ^ 0x02;
        WAITT(1);
          SIGNAL(2);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 2 'lights2':                                                    */
/*****************************************************************************/
void lights2 (void){                 /* LED operation       */
       while(1){
              WAITS(255);
          display=display ^ 0x04;
        WAITT(1);
          SIGNAL(3);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 3 'lights3':                                                    */
/*****************************************************************************/
void lights3 (void){                 /* LED operation       */
       while(1){
              WAITS(255);
          display=display ^ 0x08;
        WAITT(1);
          SIGNAL(4);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 4 'lights4':                                                    */
/*****************************************************************************/
void lights4 (void){                 /* LED operation       */
       while(1){
              WAITS(255);
           display=display ^ 0x10;
        WAITT(1);
           SIGNAL(5);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 5 'lights5':                                                    */
/*****************************************************************************/
void lights5 (void){                 /* LED operation       */
```

```
        while(1){
              WAITS(255);
          display = display ^ 0x20;
        WAITT(1);
          SIGNAL(6);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 6 'lights6':                                                    */
/*****************************************************************************/
void lights6 (void){                  /* LED operation      */
        while(1){
              WAITS(255);
          display = display ^ 0x40;
        WAITT(1);
          SIGNAL(7);
              }
 }
/*****************************************************************************/
/*****************************************************************************/
/*      Task 7 'lights7':                                                    */
/*****************************************************************************/
void lights7 (void){                  /* LED operation      */
 while(1){
       WAITS(255);
          display=display ^ 0x80;
```

```
        WAITT(1);
          SIGNAL(0);
              }
 }
/****************************************************************************/
/****************************************************************************/
/*      Task 8 'lights8':                                                   */
/****************************************************************************/
void lights8 (void){                /* LED operation      */
        while(1){
              WAITI(0);             /* wait for INTERRUPT EXT0    */
          display=0;
              }
 }
/****************************************************************************/
/****************************************************************************/
/*      Task 9 'lights9':                                                   */
/****************************************************************************/
void lights9 (void){                /* LED operation      */
       while(1){
              WAITI(2);                    /* wait for INTERRUPT EXT1       */
          display=0XFF;
              }
 }
/****************************************************************************/

/****************************************************************************/
/*       Main: Initialise and display                                       */
/****************************************************************************/
void main (void)                    { /* program execution starts here      */
  INIT_8255(0x91);                         /* initialise 8255 port          */
  INIT_RTOS(0x25);          /* initialise RTOS variables and stack          */
                            /* using timer2 and ext0 & ext1 interrupts      */
  CREATE(0,lights0);        /* start lights task                            */
  CREATE(1,lights1);        /* start lights task                            */
  CREATE(2,lights2);        /* start lights task                            */
  CREATE(3,lights3);        /* start lights task                            */
  CREATE(4,lights4);        /* start lights task                            */
  CREATE(5,lights5);        /* start lights task                            */
  CREATE(6,lights6);        /* start lights task                            */
  CREATE(7,lights7);        /* start lights task                            */
  CREATE(8,lights8);               /* start lights task                     */
  CREATE(9,lights9);               /* start lights task                     */

  display = 0;

  RTOSGOMSEC(25,0); /* start RTOS system */

  while (1){
  LEDS=display;
  }
}
/****************************************************************************/
```

Random Example Program

```
/*
*************************************************************************************
*                                            PAULOS
*                                     The Real-Time Kernel
*
*
*                                      EXAMPLE random06.c
*************************************************************************************
*/
#include <reg52.h>            /* special function registers 8052           */
#include "..\Headers\PaulosV5B.h"            /* PaulOS C version system calls definitions */
#include <absacc.h>
#include <stdio.h>
#include <stdlib.h>
#include "..\Headers\SerP2Pkg.h"
#include "..\Headers\Flt32Pkg.h"
#include "..\Headers\HYPER_PC.H"

/*
*************************************************************************************
*                                            TASKS
*************************************************************************************
*/

void CommonTask (void)
{
        uchar x,y,z,s[3];

        z = 1 + RUNNING_TASK_ID();
        PERIODICA(0,0,z);             /* Run every (1 + Task ID) seconds */
while(1)
        {
            x = rand()%80;       /* Get X position (0-79) where task number will appear */
            y = 5 + rand()%16;   /* Get Y position (5-20) where task number will appear */
            z = RUNNING_TASK_ID();
            PC_DispChar(y,x,z+'A');           /* Display the task number on the screen */
            WritePort('B',z);
            sprintf(s,"%02bu",z);
            PC_DispStr(22,40,s);
               WAITV();
        }
}
/*
*************************************************************************************
*/


/*
*************************************************************************************
*/
void ClearArea (void)
{
        uchar i,s[3];
        PERIODICA(0,0,25);                    /* Repeat every 25 seconds */
```

```
while(1)
   {
        i = RUNNING_TASK_ID();
        sprintf(s,"%02bu",i);
        PC_DispStr(22,40,s);          /* Display the task number on the screen     */
        WritePort('B', i);
        for (i=5;i<=20;i++) PC_DispClr2EndOfRow(i,0);
        PC_DispStr(22,40,s);          /* Display the task number on the screen     */
        WAITV();
   }
}

/*
**************************************************************************************
*/
void RandomSeed (void)
{
   uint x;
   uchar z,s[3];

   PERIODICA(0,0,4);                          /* Run every 4 seconds */
while(1)
   {
        z = RUNNING_TASK_ID();
        sprintf(s,"%02bu",z);
        PC_DispStr(22,40,s);          /* Display the task number on the screen */
        WritePort('B',z);
        x = (x+1)%0xFFFF;
                srand(x);
        WAITV();
   }
}

/*
**************************************************************************************
*/
/*
**************************************************************************************
*/
/*$PAGE*/
/*
**************************************************************************************
*/
/*
**************************************************************************************
*                                             MAIN
**************************************************************************************
*/
/* Using ANSI.SYS Escape control sequence             */
/* Clear Screen               Esc[2J                  */
/* Position Cursor            Esc[row;colH            */
/* Clear to end of line Esc[K                         */

void main (void)
{
```

```
    uchar i;
    INIT_RTOS(0x20);                    /* initialise RTOS variables and stack */
        Init_8255(0x91);                    /* Initialise the 8255 */
    Set_P2_BaudRate (38400);

    PC_DispClrScr();                    /* Clear the screen                  */
    PC_DispStrCntr (1,"PaulOS, The Real-Time 8051 Co-Operative Kernel");
    PC_DispStrCntr (2,"by Paul P. Debono - EXAMPLE Random 06 with 35 tasks");
    PC_DispStrCntr (3,"Version 5B");
    PC_DispStr(22,31,"Task No:");

        for(i=0;i<=32;i++)
         {
        CREATE(i,CommonTask);          /* CREATE common tasks                      */
         }

        CREATE(33,ClearArea);          /* CREATE task                              */
        CREATE(34,RandomSeed);         /* CREATE task */

        RTOSGOMSEC(250,0);             /* Start multitasking                       */

        while (1)
        {
         SET_IDLE_MODE();/* Go to idle mode if doing nothing, to conserve energy   */
        }
}

/*
*************************************************************************************
*************************************************************************************
```

Multi-Processor Example Program

This program is a Master-Slaves example, running under interrupts (not under RTOS). See remarks on the program header for the program details.

```
/* Multi-Processor Test program number one */
/* Master.c */

/* Use in conjunction with Slave.c */

/* Please include the following with Source Group */
/*                            STARTUP.A51                          */
/*                            SerP2Pkg.c                           */
/*                            Flt32Pkg.c                           */

// Main routine simply waits for keyboard
// input to send messages to slaves
// Sends any message to any slave or a General Message to all slaves
// Receives a message from the slave in communication
//  Internal Serial port handles communications (not under interrupt control)
//     to master microcontroller (socket P3 - up to 345600 baud).
//  Main program prints messages sent/received on the terminal
//     screen via socket P2, 38400 baud.

// Timer 2 interrupt service routine toggles the upper 4 leds

#include <reg52.h>
#include <absacc.h>
#include <stdio.h>
#include <string.h>

#define EOT '~'                       // END OF MESSAGE
#define EOS '\0'                      // END OF STRING

// External Functions in SerP2Pkg.c
void P2_SetUp(void);
void Auto_P2_BaudRate(void);
void Set_P2_BaudRate(unsigned int baud);
char putchar (char c);
char _getkey (void);

// External Functions if using Flt32Pkg.c
void Init_8255(unsigned char c);
void WritePort(unsigned char c, unsigned char d) large reentrant;
unsigned char ReadPort(unsigned char c) large reentrant;
//

// Functions in this module
void Init_TIMER2 (unsigned char msecs);
void XchgInfo(unsigned char SlaveNum, char s[], char r[]);
void Init_P3UART (unsigned char mode, unsigned long baudrate);

// Variables
unsigned char data intdisplay;
unsigned char data i=1;

void Init_TIMER2 (unsigned char msecs)       // Timer 2 initialisation
                                             //causes interrupt every msecs
{
```

```
unsigned char data THIGH,TLOW;
unsigned int data clock;

clock = msecs*922;                    // 922 = 11059/12;
THIGH = (65536-clock)/256;
TLOW = (65536-clock)%256;

   RCAP2H = THIGH;                    // Re-load values
   RCAP2L = TLOW;
   TH2 = THIGH;                       // set up timer 2 (for Flight 32)
   TL2 = TLOW;
   T2CON = 0x0;                       // timer 2 16-bit auto-reload mode
   ET2 = 1;
   TR2 = 1;                           // START TIMER
}

void T2ISR (void) interrupt 5 using 1 // Timer 2 ISR, toggles LEDs
{
TF2 = 0;
if (i++ == 10)                        // every 10 interrupts
        {
        i = 1;
        intdisplay = intdisplay ^ 0xF0;       // toggle upper 4 bits (^ => XOR)
        WritePort('B',intdisplay);    /* output to leds */
        }
}

void Init_P3UART (unsigned char mode, unsigned long baudrate)
{                     /* Initialise 8051 UART for multi-processor comms */

ES = 0;               /* Disable Serial Interrupt */
SCON = mode;          /* Setup serial port control register */
PCON &= 0x7F;         /* Clear SMOD bit in power ctrl reg PCON,(no double brate) */

switch (baudrate) {
   case 1200:
        TH1 = TL1 = 0xE8;
        break;
   case 2400:
        TH1 = TL1 = 0xF4;
        break;
   case 4800:
        TH1 = TL1 = 0xFA;
        break;
   case 9600:
        TH1 = TL1 = 0xFD;
        break;
   case 19200:
        TH1 = TL1 = 0xFD;
        PCON |= 0x80;         /* double baudrate, SMOD = 1 */
        break;
   case 57600:
        TH1 = TL1 = 0xFF;     /* Not quite standard */
        PCON |= 0x80;         /* double baudrate, SMOD = 1 */
        break;
```

```
   case 172800:                /* not strictly speaking standard */
        break;                 /* crystal/64, SMOD = 0 */
 case 345600:                  /* not strictly speaking standard */
        PCON |= 0x80;          /* crystal/32, SMOD = 1 */
        break;
 }

if (baudrate <= 57600)
   {
   TR1 = 1;     /* Start timer 1 (baud rate) */
   ET1 = 0;             /* Disable Timer 1 Interrupts just in case */
   TMOD &= 0x0F;        /* Setup timer/counter mode register */
                        /* Clear M1 and M0 for timer 1 */
   TMOD |= 0x20;        /* Set M1 for 8-bit auto-reload timer 1 */
   RCLK = 0;            /* USE TIMER 1 FOR RECEIVE BAUD RATE (8032 only) */
   TCLK = 0;            /* USE TIMER 1 FOR TRANSMIT BAUD RATE (8032 only) */
   TI = 1;              /* Set TI to indicate initially ready to transmit */
    }
}

void XchgInfo(unsigned char SlaveNum, char s[], char r[])
 {                                     // Sends address/data to a particular slave
                                               // Receives data from addressed slave
 unsigned int data inptr,outptr;
        unsigned char data c, x;

        while(!TI){}           // wait for any previous transmission to finish
                                               // TI = 1, means ready to load new character in
```

```
SBUF for Tx
        TB8 = 1;                // set bit 8, for address transmission
        TI = 0;                        // clr TI since we are going to transmit again
        SBUF = SlaveNum;        // send slave address
        while(!TI){}            // wait for address to be transmitted

/* This might cause problems, since program might wait here indefinitely if */
/* there is any problem in the network - needs timeout capability */

         TB8 = outptr = 0;      // address sent, now set for data transmission
         for(x=0;x<100;x++){}  // short delay to give time to affected slaves to switch
                                              // over from address to data reception mode
         while (s[outptr]!='\0')
          {
        TI = 0;                 // clr TI since we are going to transmit again
          SBUF = s[outptr++];
          while(!TI){}          // wait for byte to be transmitted
          for(x=0;x<100;x++){}          // short delay depending on receiving device
                                // may be unnecessary
          }

    inptr = 0;
        do                      // Received string expected to end with '~' character
          {
          while(!RI){}          // wait for character to be received from slave
                                // RI = 1, means data received
          RI = 0;                      // clear RI to wait for next character
          c = SBUF;                    // read data character sent from slave
          r[inptr++] = c;       // and store it
          }
        while (c != '~');       /* loop again until end of data marker '~' */
    r[--inptr] = '\0';          // overwrite received '~' with '\0'
                                // In C a string finishes with a '\0'
}

void Send2All(char s[]) /* Send a message to ALL slaves - no response expected back */
    {
    unsigned int data outptr;
        unsigned char data x;

        while(!TI){}            // wait for any previous transmission to finish
                                // TI = 1, means ready to load new character in SBUF
        TB8 = 1;                   // set bit 8, for address transmission
    TI = 0; // clear TI flag since we are goling to transmit again
        SBUF = 255;             // send General broadcast slave address (255)
        while(!TI){}            // wait for address to be transmitted

/* This might cause problems, since program might wait here indefinitely if */
/* there is any problem in the network - needs timeout capability */

        TB8 = outptr = 0;              // address sent, now set for data transmission
        for(x=0;x<200;x++){}           // short delay to give time for slaves to check address
                                       // and switch over from address to data reception mode
        while (s[outptr]!='\0')
          {
            TI = 0;
```

```
            SBUF = s[outptr++];
            while(!TI){}               // wait for byte to be transmitted
        for(x=0;x<100;x++){}
          /* just a delay for receiving device,
            since we are not using any handshaking */
          }
}

void main(void)
{
        unsigned int Msg[256],i;      // 256 slaves maximum
        unsigned char data SlaveNum;
        unsigned char Outgoing[350], Incoming[350];
// 350 characters per message maximum

// Initialise all arrays
    for (i=0;i<256;i++)
        Msg[i] = 1;                   // All start wih message 1

    for (i=0;i<350;i++)
        {
        Outgoing[i] = 0;
           Incoming[i] = 0;
           }

Set_P2_BaudRate(38400);       // initialise 2SC6911 (P2 socket) for screen and keyboard

// Init_P3UART(0xDA,57600);// initialise UART(P3 socket) for multi-processor comms
/* Setup serial port control register SCON = 0xDA, not under interrupt control */
/* Mode 3: 9-bit uart, 57600 baud rate */
/* SM0=1, SM1=1, SM2 = 0, REN = 1 */
/* TB8 = 1, RB8 = 0, TI = 1, RI = 0 */

Init_P3UART(0x9A,345600);// initialise UART(P3 socket) for multi-processor comms
/* Setup serial port control register SCON = 0x9A, not under interrupt control */
/* Mode 2: 9-bit uart, FIXED 345600 baud rate */
/* SM0=1, SM1=0, SM2 = 0, REN = 1 */
/* TB8 = 1, RB8 = 0, TI = 1, RI = 0 */

    Init_8255(0x91); // initialise 8255 input/output port
    intdisplay = 0;
    Init_TIMER2(50); /* timer 2 interrupt running every 50 milliseconds */
    EA=1; /* enable global interrupts */
    printf("\n\rThis is the MASTER controller\n\r");
    printf("\n\rMake sure you use the _getkey (wait for key) function in SerP2Pkg\n\r");
    printf("\n\rLEDs flashing under Timer 2 interrupt.\n\r");
    while (1)
        {      /* loop forever */
        printf("\rValid Slave numbers are:\n\r");
        printf("       0 - 254 for Individual commands\n\r");
        printf("       255     for General Broadcast\n\r");
         printf("Enter Slave no: ");
        scanf("%bu",&SlaveNum);
        getchar();// eliminate carriage return used for entering slave number

        if (SlaveNum != 255)          // talking to just one slave
        {
```

```
        printf("\n\rEnter message %u for slave number %bu :\n\r",
                Msg[SlaveNum]++,SlaveNum);
        gets (Outgoing, sizeof (Outgoing));
        strcat (Outgoing, "~");
        printf("\n\r   Sending the following data\n\r(%s)\n\r       to slave number: %bu\n\
r",Outgoing,SlaveNum);
        XchgInfo(SlaveNum,Outgoing,Incoming);
    // new data read from slave saved in Incoming
        printf("\n\rSlave number %bu replied :\n\r %s\n\n\n\r",SlaveNum,Incoming);
                   }

                else if (SlaveNum == 255)             // General Broadcast Message
                   {
                   printf("\n\rEnter General Broadcast Message %u for All slaves:-
\n\r",Msg[SlaveNum]++);
                gets (Outgoing, sizeof (Outgoing));
                strcat (Outgoing, "~");       /* attach end of data marker */
                   printf("\n\rSending data to ALL slaves\n\r");
                   printf(" (%s)\n\n\r",Outgoing);
                   Send2All(Outgoing);                // send data to all slaves
                   }
    }
}
/* Multi-Processor Slave Test program - NO RTOS */
/* Slave.c */

/* Use in conjunction with Master.c */
```

```
/* Please include the following with Source Group */
/*                          STARTUP.A51                          */
/*                          SerP2Pkg.c                           */
/*                          Flt32Pkg.c                           */

// Timer 2 interrupt toggles the upper 4 bits every 25 msecs.
// Internal Serial port interrupt handles communications
//      to master microcontroller (socket P3 - up to 345600 baud).
// Main program prints messages sent/received on the terminal
//      screen via socket P2, 38400 baud.

#include <reg52.h>
#include <absacc.h>
#include <stdio.h>
#include <string.h>

#define EOT '~'                        // END OF MESSAGE CHARACTER
#define EOS '\0'                       // END OF STRING CHARACTER

// External Functions in SerP2Pkg.c
void P2_SetUp(void);
void Auto_P2_BaudRate(void);
void Set_P2_BaudRate(unsigned int baud);
char putchar (char c);
char _getkey (void);

// External Functions if using Flt32Pkg.c
void Init_8255(unsigned char c);
void WritePort(unsigned char c, unsigned char d) large reentrant;
unsigned char ReadPort(unsigned char c) large reentrant;
//

// Functions in this module
void Init_TIMER2 (unsigned char msecs);
void Init_P3UART_int (unsigned char mode, unsigned long baudrate);
void Byte2Bin (unsigned char CH, char s[]);

// Variables
bit MasterCalled, Its4Me, Its4All;
unsigned char data i;
unsigned int data inptr,outptr;
unsigned char data intdisplay, ItIsMe, Broadcast;
unsigned char RxString[250], TxString[250];

void Init_P3UART_int (unsigned char mode, unsigned long baudrate)
// Set up internal UART (P3) under interrupt control
{
SCON = mode;          /* Setup serial port control register */
PCON &= 0x7F;         /* Clear SMOD bit in power ctrl reg */

switch (baudrate) {   // set up timer 1 initial count according to baud rate required
   case 1200:
        TH1 = TL1 = 0xE8;
        break;
   case 2400:
        TH1 = TL1 = 0xF4;
        break;
```

```
    case 4800:
         TH1 = TL1 = 0xFA;
         break;
    case 9600:
         TH1 = TL1 = 0xFD;
         break;
    case 19200:
         TH1 = TL1 = 0xFD;
         PCON |= 0x80;          /* double baudrate, SMOD = 1 */
         break;
    case 57600:
         TH1 = TL1 = 0xFF;      /* Not quite standard */
         PCON |= 0x80;          /* double baudrate, SMOD = 1 */
         break;
    case 172800:                /* not strictly speaking standard */
         break;                 /* crystal/64, SMOD = 0 */
    case 345600:                /* not strictly speaking standard */
         PCON |= 0x80;          /* crystal/32, SMOD = 1 */
         break;
 }

if (baudrate <= 56700)
  {
    ET1 = 0;
    TMOD &= 0x0F;               /* Setup timer/counter mode register */
                                /* Clear M1 and M0 for timer 1 */
    TMOD |= 0x20;               /* Set M1 for 8-bit autoreload timer 1 */
    RCLK = 0;                   /* USE TIMER 1 FOR RECEIVE BAUD RATE (8032 only) */
    TCLK = 0;                   /* USE TIMER 1 FOR TRANSMIT BAUD RATE (8032 only) */
    TR1 = 1;                    /* Start timer 1 (baud rate) */

    TI = 0;                     /* Clear TI to indicate not ready to xmit yet */
    ES = 1;                     /* Enable Serial Interrupt */
  }
}

void Init_TIMER2 (unsigned char msecs)       // Timer 2 initialisation
{
unsigned char data THIGH,TLOW;
unsigned int data clock;

clock = msecs*922;
// 922 = 11059/12 = counts required for 1 msec TF2 interrupt delays;
THIGH = (65536-clock)/256;
TLOW = (65536-clock)%256;

    RCAP2H = THIGH;
    RCAP2L = TLOW;
    TH2 = RCAP2H;               // set up timer 2 (for Flight 32)
    TL2 = RCAP2L;
    T2CON = 0x0;                // timer 2 16-bit auto-reload mode
    ET2 = 1;
    TR2 = 1;                    // START TIMER 2
}
```

```
void P3uart_isr (void) interrupt 4 using 1   /* Runs as a Serial Interrupt Routine */
/* sends TxString and receives RxString */
{
unsigned char data c;
unsigned int x;
// interrupt may be caused either from a TI or an RI flag

// RECEIVER SECTION
if(RI)                                 // if a character is received
    {
    RI = 0;                            // reset flag
    c = SBUF;                          // get character
    if (RB8==1 && c==ItIsMe)           // if received correct address,
       {
          SM2 = 0;                                     // prepare to read data
       RB8 = 0;
          Its4All = 0;
          Its4Me = 1;
          inptr = 0;
    }

 else if (RB8==1 && c==Broadcast) // if received general broadcast address,
    {
       SM2 = 0;                 // prepare to read data
    RB8 = 0;
    Its4All = 1;
       Its4Me = 0;
       inptr = 0;
    }
```

```
 else if ((SM2==0) && (c != '~'))     // store any received data
    RxString[inptr++] = c;
 else if ((SM2==0) && (c == '~'))
    {
    MasterCalled = 1;
    RxString[inptr] = '\0';           // end received string with NULL character
       SM2 = 1;                       // reset serial port for address reception mode
    }
 }

// TRANSMITTER SECTION
  if(TI && (TxString[outptr] != '\0')) // just sends a message from TxString
    {
    TI = 0;                           // clr TI in order to transmit
       for (x=1;x<500;x++){};
  /* just a delay for receiving no handshaking */
    SBUF = TxString[outptr++];
    }

  else if(TI && (TxString[outptr] == '\0')) // No more data to send
    TI = outptr = 0;
}

void T2ISR (void) interrupt 5 using 2        // Timer 2 interrupt service routine
{
TF2 = 0;
if (i++ == 5)                                // every 5 interrupts
       {
       i = 1;
       intdisplay = intdisplay ^ 0xF0; // toggle upper 4 bits (XOR)
    WritePort('B',intdisplay); /* output to leds */
       }
}

void Byte2Bin (unsigned char CH, char s[])
{
       unsigned char data i, c=CH;
       unsigned char data Mask = 1<<7;

       for (i=1;i<=8;i++)
         {
           s[i-1] = (c & Mask ? '1' : '0');
           c <<= 1;
 }
}

void main(void)
{
       unsigned int data MsgNo = 1; // start with first message
       unsigned char data SwitchSettings, s[8];
       unsigned char SwitchString[80];

       Init_8255(0x91);               // Initilaise 8255 PIO
       Set_P2_BaudRate(38400);        // initialise external UART (P2) for keyboard/screen
```

```
//Init_P3UART_int(0xF0,57600);
// initialise internal UART (P3 socket) for multi-processor comms (interrupt)
/* Setup serial port control register SCON = 0xF0 */
/* Mode 3: 9-bit uart var. baud rate */
/* SM0 = SM1 = SM2 = REN = 1 */
/* TB8 = RB8 = TI = RI = 0 */
/* under interrupt control */

Init_P3UART_int(0xB0,345600);
// initialise internal UART (P3 socket) for multi-processor comms (interrupt)
/* Setup serial port control register SCON = 0xB0 */
/* Mode 2: 9-bit uart FIXED baud rate */
/* SM0 = 1, SM1 = 0, SM2 = REN = 1 */
/* TB8 = RB8 = TI = RI = 0 */
/* under interrupt control */

        intdisplay = 0;         // clear display
        Init_TIMER2(50);        /* timer 2 interrupt running every 50 milliseconds */
        EA=1;                           /* enable global interrupts */

        Broadcast = 255;        /* General Broadcast address - all slaves receive this */
            printf("\n\rHello, please enter a Unique number for this Slave (0-254) : ");
            scanf("%bu",&ItIsMe);
        printf("\n\rEntering main loop (leds flashing under interrupt Timer 2),\n\rwhich will be
interrupted\n\n\r");
        printf("EITHER\n\r");
        printf(" (i) by a message dedicated only to this slave number: %bu\n\r",ItIsMe);
            printf("OR\n\r");
        printf("(ii) by a Global Broadcast Message.\n\n\n\r");

            inptr = outptr = 0;         // variables used to scan TxString and RxString
        while (1)
            {           /* loop forever */

        if(MasterCalled && Its4Me) // message is just for me
               {
               MasterCalled = Its4Me = 0;
               /* Start Timer 2 interrupts */
               /* They may have been switch off by a Global Message */
               ET2 = 1;
               TR2 = 1;
               /* Get lower four-bit switch settings */
               SwitchSettings = ReadPort('A') & 0x0F;
               Byte2Bin(SwitchSettings,s);
               sprintf(TxString,
                  "Hello, this is Slave no: %bu, acknowledging your message no: %bu.%u\n\r",
                       ItIsMe,ItIsMe,MsgNo);
               sprintf(SwitchString," Lower 4-bit switch settings are: %#bx HEXADECIMAL or %s
BINARY.\n\n\n\r",
                       SwitchSettings, s);
               strcat(TxString,SwitchString);
               // Message must end with the '~' character
               strcat(TxString,"~");
```

```
          printf("Received from Master, message no: %u.\n\r%s\n\r",MsgNo++,RxString);
               printf("Now sending to Master this acknowledgement:\n\r (%s)\n\n\n\
r",TxString);

               TI = 1; // send TxString message (via interrupt routine, UART P3)
          }

        else if(MasterCalled && Its4All)
               {
               MasterCalled = Its4All = 0;
          printf("Received from Master, Global Broadcast message no: %u.\n\r%s\n\
r",MsgNo++,RxString);
               printf(" Switching off LED flashing routine.\n\r");
                  ET2=0;
                     TR2=0;
                     WritePort('B',0);
               printf("  This message is not acknowledged back to the Master\n\r");
               printf("  since all the slaves would be doing it at the same time\n\r");
               printf("  and the data would therefore be corrupted.\n\n\n\r");
        }
          }
}
/* Multi-Processor Slave Test program */
/* SlaveRtosDemo4.c */

/* Compatible with PaulOS RTOS */

/* 7 TASKS */
```

```
/* Disable PERIODIC and enable HALFMSEC in TaskStkV5B.A51 */

/* Use in conjunction with MasterRtosDemo4.c */

/* Please include the following with Source Group */
/*                           STARTUP.A51                          */
/*                           PaulOS.A51                           */
/*                           SerP2Pkg.c                           */
/*                           Flt32Pkg.c                           */

// One task toggles the lower 4 bits every 25 msecs.
// Other tasks control the motor speed which is set by the master controller
// Serial port interrupt handles coms to master microcontroller
// Main program prints messages sent/received on the terminal screen

#include <reg52.h>
#include <absacc.h>
#include <stdio.h>
#include <string.h>
#include "..\Headers\PaulosV5B.h"            /* PaulOS RTOS system calls definitions */

// EOM '~'                     // END OF MESSAGE
// EOS '\0'                    // END OF STRING

// External Functions in SerP2Pkg.c
void P2_SetUp(void);
void Auto_P2_BaudRate(void);
void Set_P2_BaudRate(unsigned int baud);
char putchar (char c);
char _getkey (void);

// External Functions if using Flt32Pkg.c
void Init_8255(unsigned char c);
void WritePort(unsigned char c, unsigned char d) large reentrant;
unsigned char ReadPort(unsigned char c) large reentrant;
//

// Functions in this module
void init_TIMER2 (unsigned char msecs);
void init_P3UART_int (unsigned char mode, unsigned int baudrate);

// Variables
bit MasterCalled, Its4Me, Its4All, MotorOK;
unsigned int data inptr, outptr;
unsigned char data ItIsMe, setting, speed, Broadcast;
unsigned char RxString[250], TxString[250], TempString[50];
unsigned int data MsgNo; // start with first message
unsigned char bdata display;
sbit MOTOR = display^7;

void Byte2Binary (unsigned char x, char b[])
// Convert a byte to its binary value as an ASCII string of 1's and 0's stored in b[]
{
unsigned char i;
unsigned char Mask = 1 << 7;
for (i=0; i <= 7; i++)
           {
           b[i] = (x & Mask ? '1' : '0');
```

```
        x <<= 1;
        }
        b[8] = '\0';
}

void init_P3UART_int (unsigned char mode, unsigned int baudrate)
// Set up internal UART (P3) under interrupt control
{
SCON = mode;           /* Setup serial port control register */
PCON &= 0x7F;          /* Clear SMOD bit in power ctrl reg - No double baud rate yet */
TMOD &= 0x0F;          /* Setup timer/counter mode register */
                       /* Clear M1 and M0 for timer 1 */
TMOD |= 0x20;          /* Set M1 for 8-bit autoreload timer 1 */
RCLK = 0;              /* USE TIMER 1 FOR RECEIVE BAUD RATE (8032 only) */
TCLK = 0;              /* USE TIMER 1 FOR TRANSMIT BAUD RATE (8032 only) */
                       /* TH1 = 256 - (28800/BR) */
switch (baudrate) {    // set up timer 1 initial count according to baud rate required
    case 300:
       TH1 = TL1 = 0xA0;
       break;
    case 600:
       TH1 = TL1 = 0xD0;
       break;
    case 1200:
       TH1 = TL1 = 0xE8;
       break;
    case 2400:
       TH1 = TL1 = 0xF4;
       break;
    case 4800:
       TH1 = TL1 = 0xFA;
       break;
    case 9600:
       TH1 = TL1 = 0xFD;
       break;
    case 19200:
       TH1 = TL1 = 0xFD;
       PCON |= 0x80;          /* double baudrate (SMOD = 1) */
       break;
    case 57600:
       TH1 = TL1 = 0xFF;
       PCON |= 0x80;          /* double baudrate (SMOD = 1) */
       break;
    }

ET1 = 0;               /* Disable timer 1 interrupts just in case */
TR1 = 1;               /* Start timer 1 (baud rate) */
TI = 0;                /* Clear TI to indicate not ready to xmit */
                       /* Serial Interrupt enabled by the RTOS */
}

// TASK 0
/* Sends TxString to Master and receives RxString from Master */

void P3uart_isr(void)
{
```

```
unsigned char data c,x;
while (1)
{
WAITI(4);      // wait for a serial interrupt
// interrupt may be caused either from a TI or an RI flag

// RECEIVER SECTION
if(RI)                               // if a character is received
   {
   RI = 0;                           // reset flag
   c = SBUF;                         // get character
   if (RB8==1 && c==ItIsMe)          // if received correct address,
       {
       SM2 = 0;               // prepare to read data
   RB8 = 0;
       Its4All = 0;
       Its4Me = 1;
       inptr = 0;
   }
else if (RB8==1 && c==Broadcast) // if received general broadcast address,
   {
       SM2 = 0;                      // prepare to read data
   RB8 = 0;
   Its4All = 1;
       Its4Me = 0;
       inptr = 0;
   }
```

```
else if ((SM2==0) && (c != '~')) // store any received data
   RxString[inptr++] = c;
else if ((SM2==0) && (c == '~')) // end of received message with ~ character
   {
   RxString[inptr] = '\0'; // add to end of received message a NULL character
       SM2 = 1; // reset for address reception mode
   MasterCalled = 1; // Set flag to indicate message received
   if (Its4Me==1 && MasterCalled==1)
       SIGNAL(2); // Activate Task 2 - Private Message Reception
   else if (Its4All==1 && MasterCalled==1)
       SIGNAL(3); // Activate Task 3 - Global Message Reception
       }
}

// TRANSMITTER SECTION
  if(TI && (TxString[outptr] != '\0'))       // just sends a message from TxString
     {
     TI = 0;                          // clr TI in order to transmit
     SBUF = TxString[outptr++];              // TI will be set to 1, once transmission is ready
     for (x=0;x<10;x++){}                    // just a delay for receiving equipment
     }

  else if(TI && (TxString[outptr] == '\0')) // No more data to send
     TI = outptr = 0;
}
}

// TASK 1

void Blinker (void)    // Blinks Leds
{
while(1)
     {
     display ^= 0x0F;
     WritePort('B',display); /* output to leds toggle lower 4 bits (XOR)*/
     WAITT(500);
       }
}

/*****************************************************************************/
// TASK 2
// Private Message Data Reception and Acknowledgement
/*****************************************************************************/

void PrivateDataRx (void)
{
while (1)
       {
       WAITS(0);                      // Wait for signal indefinitely
          MasterCalled = 0;
       printf("Rx Msg: %u <== %s\n\r",MsgNo++,RxString);

       if (strncmp ("Speed", RxString, 5)==0)
                     {
              sscanf(RxString,"%*s %bu",&setting); // read received message: 'Speed n'
                              // ignoring 'Speed' and making setting = n
                     sprintf(TxString, "OK. Speed set to %bu", setting);
                     MotorOK = 1;
                     }
```

```
        else if (strncmp ("What Speed?", RxString, 11)==0)
                        sprintf(TxString, "Current Speed setting %bu", setting);

        else if (strncmp ("LineChk", RxString, 7)==0)
                        sprintf(TxString, "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 
0123456789");

        else if (strncmp ("Stop", RxString, 4)==0)
                        {
                        sprintf(TxString,"OK. Motor stopped");
                        MotorOK = 0;
                        }

         else if (strncmp ("Switches", RxString, 8)==0)
                        {
                        sprintf(TxString, "Switches reading: ");
                        Byte2Binary(ReadPort('A'),TempString);
                        strcat(TxString,TempString);
                        }

        else
                        sprintf(TxString, "Command NOT listed. No action taken.");
        strcat(TxString,"~");
        DEFER();
        printf("Tx Msg: ==> %s \n\n\r",TxString);
        TI = 1; // if master called, THEN send TxString message
                // (via task 0, which is waiting for a serial interrupt routine,
                // activated with TI=1)
      }
}

/*****************************************************************************/
// TASK 3
// Global Message Reception
/*****************************************************************************/

void GlobalRx (void)
{
while (1)
        {
        WAITS(0);                    // Wait for signal indefinitely
           MasterCalled = 0;
        printf("Rx Global Broadcast Msg: %u.\n\r ( %s )\n\r",MsgNo++,RxString);
           printf("   This message is not acknowledged back to the Master\n\r");
           printf("   since all the slaves would be replying it at the same time\n\r");
           printf("   and the data would therefore be corrupted\n\n\n\r");

        if (strncmp ("Start", RxString, 5)==0)
                        {
           setting = 3;
                        MotorOK = 1;
                        printf("Starting motor at a speed setting of 3\n\n\n\r");
                        }

           else if (strncmp ("Resume", RxString, 6)==0)
               {
                        printf("Restarting motor at a speed setting of %bu\n\n\n\r",setting);
```

```
            MotorOK = 1;
                      }

           else if (strncmp ("Emergency", RxString, 9)==0)
               {
                      printf("EMERGENCY - Stopping motor\n\n\n\r");
            MotorOK = 0;
                      }
         else if (strncmp ("Stop", RxString, 4)==0)
               {
                      printf("STOP - Stopping motor\n\n\n\r");
            MotorOK = 0;
                      }
           }
}

/*****************************************************************************/
/*      Task 4 'Main PWM':                                                   */
/*****************************************************************************/
void MainPWM(void){           /* PWM task    */
       while(1){
// setting ranges from 1 to 10, value received from master controller
// PWM period is going to be 101, hence maximum ON time is being limited to 100
       speed = 10*setting;
              SIGNAL(5);            /* send signal to task 5 - Motor ON */
              WAITT(101);           /* wait for timeout (main pwm period) */
              }
}

/*****************************************************************************/
/*      Task 5 'MOTOR ON':                                                   */
/*****************************************************************************/
void MotorON (void){          /* switch on motor task */
       while(1){
              WAITS(0);      /* wait for signal indefinitely */

        if (MotorOK)
                      {
                      MOTOR = 1;
                      WritePort('B',display);              /* switch on motor */
              WAITT(speed);         /* wait for specified timeout */
                 }
        SIGNAL(6);                          /* send signal to task 6 - Motor OFF */
              }
 }

/*****************************************************************************/
/*      Task 6 'MOTOR OFF':                                                  */
/*****************************************************************************/
void MotorOFF (void){         /* switch off MOTOR task */
       while(1){
              WAITS(0);      /* wait for signal indefinitely */
              MOTOR = 0;
       WritePort('B',display);       /* switch off motor */
              }
}

/****************************************************************************/

void main(void)
{
```

```
unsigned int brate,i;

        Init_8255(0x91);              // Initilaise 8255 PIO
      Set_P2_BaudRate(38400);         // initialise external UART (P2) for keyboard/screen
        display = 0;                  // clear leds
      setting = 5;
      Broadcast = 255;        // Address used to send message to ALL slaves
      MsgNo = 1;
      MasterCalled = 0;
        for (i=0;i<250;i++)
        {
        TxString[i] = 0;
            RxString[i] = 0;
            }
        for (i=0;i<50;i++)
        TempString[i] = 0;

        printf("\n\n\rWelcome to the Master-SLAVE PaulOS RTOS Demo Program 4\n\r");
        printf("Input the LINK baud rate required: ");
        scanf ("%u",&brate);
        printf("\n\r   8051 Microcontroller link running at %u baud\n\n\n\r",brate);
        printf("    Written by Paul P. Debono - November 2003.\n\n\n\r");
        init_P3UART_int(0xF0,brate);
              // initialise UART(P3 socket) for multi-processor comms
                    /* Setup serial port control register SCON = 0xF0 */
                    /* Mode 3: 9-bit uart var. at specified baud rate */
                          /* SM2, REN set to 1, TB8 = RB8 = 0 */
        INIT_RTOS(0x10); /* initialise RTOS,variables, stack with T2 + Serial int. */
              /* this must be the first RTOS command to be executed */
   CREATE(0,P3uart_isr);      /* Tx - Rx UART task */
   CREATE(1,Blinker);         /* Flash LEDs task */
   CREATE(2,PrivateDataRx);   /* Decode received private message task */
   CREATE(3,GlobalRx);        /* Decode received global message task */
   CREATE(4,MainPWM);         /* MAIN PWM TASK              */
   CREATE(5,MotorON);         /* MOTOR ON task              */
   CREATE(6,MotorOFF);        /* MOTOR OFF task             */

        printf("\n\r   Please enter a Unique number for this slave (0-254) : ");
        scanf("%bu",&ItIsMe);
        printf("\n\r      Leds flashing independently under Task 1\n\n\r");
   printf("\n\rEntering main loop, which will be interrupted\n\r");
        printf("(apart from the RTOS tick time interrupt),\n\n\r");
   printf("EITHER\n\r");
   printf(" (i) by a message dedicated only to this slave number: %bu\ (Tasks 0 and 2)\n\n\
r",ItIsMe);
        printf("OR\n\r");
   printf(" (ii) by a Global Broadcast Message (Tasks 0 and 3)\n\n\n\r");

        inptr = outptr = 0;    // variables used to scan TxString and RxString
;       RTOSGOMSEC(1,1);       // Start RTOS ticking at 1 msec, with priorities enabled
        RTOSGOMSEC(1,0);// Start RTOS ticking at 1 msec, with priorities disabled
   while (1)
        {
        SET_IDLE_MODE();               /* loop forever here, going to idle every time */
                                   /* Awake only for any interrupt */
        }
}
```

Appendix C SanctOS.C

This is the source listing for the SanctOS (Small ANd CompacT Operating System) round-robin operating system written in C. It is practically the C-version of ParrOS.A51 found in Appendix A.

It consists of

- The header file Parameters.h
- The assembly language include file SanctOS_A01.a51
- The header file SanctOS_V01.h
- The main RTOS program SanctOS.c

Parameters.h

```
/*
****************************************************************************
*                      PARAMETERS.H   ---   RTOS KERNEL HEADER FILE
*
* For use with SanctOS_V01.C -
* Round-robin RTOS written in C by Ing. Paul P. Debono
* for use with the 8051 family of microcontrollers
*
* File          : Parameters_V01.H
* Revision      : 8
* Date          : February 2006
* By            : Paul P. Debono
*
*                      B. Eng. (Hons.) Elec. Course
*                      University Of Malta
*
****************************************************************************
*/
/*
****************************************************************************
*                                             RTOS USER DEFINITIONS
****************************************************************************
*/
#define STACKSIZE       0x10    // Max stack size for task - no need to change
#define CPU                8032        // set to 8051 or 8032
#define TICK_TIMER      2       // Set to 0, 1 or 2 to select which timer to
                                // use as the RTOS tick timer
#define TICKTIME           50          // Length of RTOS basic tick in msec
#define NOOFTASKS          6           // Number of tasks used in the application
                                // program

/*
****************************************************************************
****************************************************************************
****************************************************************************
*/
```

SanctOS_A01.A51

```
; SanctOS_A01.A51
; SanctOS RTOS PLUS MAIN PROGRAM
;
; STORES ALL TASK REGISTERS
;
; Written by Paul P. Debono - JUNE 2006
; University of Malta
; Department of Communications and Computer Engineering
; MSIDA MSD 06; MALTA.
; Accomodates up to 255 tasks
;
; STACK MOVING VERSION - MOVES WORKING STACK IN AND OUT OF
; EXTERNAL MEMORY
; SLOWS DOWN RTOS, BUT DOES NOT RESTRICT TASK CALLS
; IDLE TASK (ENDLESS MAIN PROGRAM - TASK NUMBER = NOOFTASKS)
;
; THIS IS STILL A SMALL TEST VERSION RTOS. IT IS JUST USED FOR
; SHOWING WHAT IS NEEDED TO MAKE A SIMPLE RTOS.
; IT MIGHT STILL NEED SOME MORE FINE TUNING.
; IT HAS NOT BEEN NOT THOROUGHLY TESTED !!!!
; WORKS FINE SO FAR.
; NO RESPONSABILITY IS TAKEN.

$NOMOD51
#include "reg52.h"
#include "Parameters.h"
```

```
/* The MAINSTACK variable points to the start pointer in hardware stack */
/* and is defined in STARTUP.A51 */
extrn idata (MAINSTACK)

PUBLIC _SaveBank0, _RecallBank0
PUBLIC POP5I

; RTOS ASSEMBLY CODE MACROS

SANCTOS_ASM SEGMENT CODE
RSEG SANCTOS_ASM

POP5I:
   DEC SP       ; BLANK TO POP UNUSED RETURN ADDRESS
   DEC SP
   POP PSW
   POP DPL
   POP DPH
   POP B
   POP ACC
   SETB EA      ; re-enable interrupts
   RETI         ; JUMPS TO PREVIOUSLY PRE-EMPTIED TASK HERE

_SaveBank0:            ; Address high byte in R6, low byte in R7 bank 1
   MOV DPH,0EH
   MOV DPL,0FH
   MOV A,0
   MOVX @DPTR,A
   INC DPTR
   MOV A,1
   MOVX @DPTR,A
   INC DPTR
   MOV A,2 MOVX @DPTR,A
   INC DPTR MOV A,3
   MOVX @DPTR,A
   INC DPTR
   MOV A,4
   MOVX @DPTR,A
   INC DPTR
   MOV A,5
   MOVX @DPTR,A
   INC DPTR
   MOV A,6
   MOVX @DPTR,A
   INC DPTR
   MOV A,7
   MOVX @DPTR,A
   RET

_RecallBank0:  ; Address high byte in R6, low byte in R7 bank 1
   MOV DPH,0EH
   MOV DPL,0FH
   MOVX A,@DPTR
   MOV 0,A
   INC DPTR
   MOVX A,@DPTR
```

```
   MOV 1,A
   INC DPTR
   MOVX A,@DPTR

   MOV 2,A
   INC DPTR
   MOVX A,@DPTR
   MOV 3,A
   INC DPTR
   MOVX A,@DPTR
   MOV 4,A
   INC DPTR
   MOVX A,@DPTR
   MOV 5,A
   INC DPTR
   MOVX A,@DPTR
   MOV 6,A
   INC DPTR
   MOVX A,@DPTR
   MOV 7,A
   RET

END
```

SanctOS_V01.H

```
/*
****************************************************************
*                      RTOS KERNEL HEADER FILE
*
* For use with SanctOS_V01.C
*          Co-Operative RTOS written in C
*              by Ing. Paul P. Debono
*       Use with the 8051 family of microcontrollers
*
* File          : SanctOS_V01.H
* Revision      : 1
* Date          : February 2006
* By            : Paul P. Debono
*
*                       B. Eng. (Hons.) Elec. Course
*                       University Of Malta
*
***********************************************************
*/
#include "Parameters.H"

/*
***********************************************************
*                              DATA TYPE DEFINITIONS
***********************************************************
*/

typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned long ulong;
```

```
/*
***********************************************************
*                      STRUCTURE AND UNION DEFINITIONS
***********************************************************
*/

struct task_param {            /* 6 bytes + 8 registers + stack */
       uchar status1;          /* status flags, see details below */
       uint slot_time;         /* slot time allocated for this task */
       uint slot_reload;       /* slot time reload value */
       uchar stackptr;         /* stack pointer SP storage location */
                               /* registers storage area, */
                               /* ready for context switching */
       uchar reg0;
       uchar reg1;
       uchar reg2;
       uchar reg3;
       uchar reg4;
       uchar reg5;
       uchar reg6;
       uchar reg7;
       char stack[STACKSIZE];        /* stack storage area */
              };

/*
***************************************************************************
*                              DATA TYPE DEFINITIONS
***************************************************************************
*/

/*
***************************************************************************
*/
/* The MAINSTACK pointer variable points to the stack pointer in hardware
/* stack and should be defined in STARTUP.A51 */

extern idata unsigned char MAINSTACK[STACKSIZE];
extern data unsigned char Running;

/* Functions written in assembly language, found in SanctOS_A01.A51 */
extern void SaveBank0(uchar xdata * Pointer);
extern void RecallBank0(uchar xdata * Pointer);
extern void POP5I(void);

/*
***************************************************************************
*                                     FUNCTION PROTOTYPES
***************************************************************************
*
*
* The following RTOS system calls do not receive any parameters :
* ---------------------------------------------------------------
*/

void OS_RTOS_GO (void);       // Starts the RTOS running with
```

```
/* The following commands are simply defined as MACROS below
       OS_CPU_IDLE()          Set the microprocessor into a sleep mode
                                     (awake every interrupt)
       OS_CPU_DOWN()          Switch off microprocessor, activate only by
                                     hardware reset
       OS_PAUSE_RTOS()        Disable RTOS
       OS_RESUME_RTOS()       Re-enable RTOS
*/

/*
* The following RTOS system calls do receive parameters :
* -------------------------------------------------------
*/

void OS_INIT_RTOS (uchar iemask);     // Initialises all RTOS variables

void OS_CREATE_TASK (uchar tasknum, uint taskadd, uint slot);
                                     // Creates a task

/*
***************************************************************************
*/

/*
***************************************************************************
*                             RTOS TIMING DEFINITIONS
***************************************************************************
*/

#define MSEC10 9216UL   // In theory 921.6 counts represent
                      // 1 ms assuming an 11.0592 MHz crystal.
#define TICKS_PER_SEC   (1000 / TICKTIME)
// Ensure that TICKTIME's value is
// chosen such that this
// quotient and hence all the
// following quotients result
// in an integer. In theory, maximum
// value of TICKTIME
// is given by the value corresponding
// to CLOCK = 65535
#define TICKS_PER_MIN   (60000 / TICKTIME)
#define CLOCK   ((TICKTIME * MSEC10)/10UL)
#define BASIC_TICK      (65535 - CLOCK + 1)
// i.e. approx. 70-72 - However
// respecting the condition
// above, max. acceptable
// TICKTIME = 50 msecs. Hence all
// suitable values are:
// 1, 2, 4, 5, 8, 10, 20, 25, 40, 50

#define IDLE_TASK NOOFTASKS
// Main endless loop in application given a task
// number equal to NOOFTASKS
```

```
/* OTHER #defines */

#define MINUS_ONE       0xFF
#define ZERO            0
#define TEN             0x0A /* slot time in ticks */

#define HiByte(intval) (##intval)/256;
#define LoByte(intval) (##intval)%256;

/*
***************************************************************************
*/

/*
***************************************************************************
*                              RTOS MACROS
***************************************************************************
*/

#define OS_CPU_IDLE()          PCON |= 0x01 // Sets the microprocessor in
                                      // idle mode
#define OS_CPU_DOWN()          PCON |= 0x02 // Sets the microprocessor in
                                      // power-down mode

#if (TICK_TIMER == 0)
    #define OS_PAUSE_RTOS() EA = ET0 = TR0 = 0
    #define OS_RESUME_RTOS() TR0 = ET0 = EA = 1

#elif (TICK_TIMER == 1)
    #define OS_PAUSE_RTOS() EA = ET1 = TR1 = 0
    #define OS_RESUME_RTOS() TR1 = ET1 = EA = 1

#elif (TICK_TIMER == 2)
    #define OS_PAUSE_RTOS() EA = ET2 = TR2 = 0
    #define OS_RESUME_RTOS() TR2 = ET2 = EA = 1

#endif

/*
***************************************************************************
*/

/*
***************************************************************************
*                                      COMPILE-TIME ERROR TRAPPING
***************************************************************************
*/

#if (CPU != 8032) && (CPU != 8051)
       #error Invalid CPU Setting
#endif

#if (NOOFTASKS > 255)
       #error Number of tasks is too big. MAX 255 (from 0 to 254) tasks
#endif

#if ((TICKTIME * 110592 / 120) > 65535)
       #error Tick time value exceeds valid range for timer counter setting
```

```
#endif

#if ((TICKTIME * 110592 / 120) < 65535) && ((1000 % TICKTIME) != 0)
#error Undesirable TICKTIME setting (1, 2, 4, 8, 10, 20, 25, 40, 50 ms)
#endif

#if (CLOCK > 65535)
#error Timer counter setting exceeded valid range. Check TICKTIME and MSEC
#endif

/*
***************************************************************************
*/


/*
* Other functions used internally by the RTOS :
* ---------------------------------------------
*/

void PE_TaskChange (void);                  // Task swapping function
void RTOS_Timer_Int (void);                 // RTOS Scheduler ISR
/*
***************************************************************************
***************************************************************************
***************************************************************************
***************************************************************************
***************************************************************************
*/
```

STARTUP.A51

```
$NOMOD51
;-------------------------------------------------------------------------
;  This file is part of the C51 Compiler package
;  Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
;-------------------------------------------------------------------------
;  STARTUP.A51: This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;      A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the
;  following BL51 invocation:
;
;      BL51 <your object file list>, STARTUP.OBJ <controls>
;
;--------------------------------------------------------------------------
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;              ; the absolute start-address of IDATA memory is always 0
```

```
;IDATALEN       EQU     80H             ; the length of IDATA memory in bytes.
IDATALEN        EQU     100H            ; the length of IDATA memory in bytes for
                                          the 8032 (256 bytes).
;
;
XDATASTART      EQU     0H              ; the absolute start-address of XDATA memory
XDATALEN        EQU     0H              ; the length of XDATA memory in bytes.
;
PDATASTART      EQU     0H              ; the absolute start-address of PDATA memory
PDATALEN        EQU     0H              ; the length of PDATA memory in bytes.
;
;  Notes: The IDATA space overlaps physically the DATA and BIT areas of
;         the 8051 CPU. At minimum the memory space occupied from the C51
;         run-time routines must be set to zero.
;------------------------------------------------------------------------------
;
;  Reentrant Stack Initilization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialise it:
;
;  Stack Space for reentrant functions in the SMALL model.
IBPSTACK        EQU     1               ; set to 1 if small reentrant is used.
IBPSTACKTOP     EQU     0FFH+1          ; set top of stack to highest location+1.
;IBPSTACKTOP    EQU     07FH+1          ; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the LARGE model.
XBPSTACK        EQU     0               ; set to 1 if large reentrant is used.
XBPSTACKTOP     EQU     0FFFFH+1        ; set top of stack to highest location+1.
;
;  Stack Space for reentrant functions in the COMPACT model.
PBPSTACK        EQU     0               ; set to 1 if compact reentrant is used.
PBPSTACKTOP     EQU     0FFFFH+1        ; set top of stack to highest location+1.
;
;------------------------------------------------------------------------------
;
;  Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
;  The following EQU statements define the xdata page used for pdata
;  variables. The EQU PPAGE must conform with the PPAGE control used
;  in the linker invocation.
;
PPAGEENABLE     EQU     0       ; set to 1 if pdata object are used.
;
PPAGE           EQU     0       ; define PPAGE number.
;
PPAGE_SFR       DATA    0A0H    ; SFR that supplies uppermost address byte
;               (most 8051 variants use P2 as uppermost address byte)
;
;------------------------------------------------------------------------------
```

```
; Standard SFR Symbols
ACC     DATA    0E0H
B       DATA    0F0H
SP      DATA    81H
DPL     DATA    82H
DPH     DATA    83H

                NAME      ?C_STARTUP

?C_C51STARTUP  SEGMENT   CODE
?STACK         SEGMENT   IDATA

#include <parameters.h>
               RSEG    ?STACK
MAINSTACK:     DS      STACKSIZE

               EXTRN CODE (?C_START)
               PUBLIC ?C_STARTUP
               PUBLIC MAINSTACK

; MON51 or FLT32 should be defined in the A51 Tab in KEIL
$IF     (MON51)
               CSEG AT 8000H  ; FOR DEV BOARD MON-51 MONITOR PROG
$ELSEIF (FLT32)
               CSEG AT 8100H  ; FOR FLT-32 DEV BOARD MONITOR PROG
$ELSE
               CSEG AT 0      ; FOR EEPROM
$ENDIF

?C_STARTUP:    LJMP    STARTUP1
               RSEG    ?C_C51STARTUP

STARTUP1:
IF IDATALEN <> 0
               MOV     R0,#IDATALEN - 1
               CLR     A
IDATALOOP:     MOV     @R0,A
               DJNZ    R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
               MOV     DPTR,#XDATASTART
               MOV     R7,#LOW (XDATALEN)
   IF (LOW (XDATALEN)) <> 0
               MOV     R6,#(HIGH (XDATALEN)) +1
   ELSE
               MOV     R6,#HIGH (XDATALEN)
ENDIF
               CLR     A
XDATALOOP:     MOVX    @DPTR,A
               INC     DPTR
               DJNZ    R7,XDATALOOP
               DJNZ    R6,XDATALOOP
ENDIF
```

```
IF PPAGEENABLE <> 0
               MOV     PPAGE_SFR,#PPAGE
ENDIF

IF PDATALEN <> 0
               MOV     R0,#LOW (PDATASTART)
               MOV     R7,#LOW (PDATALEN)
               CLR     A
PDATALOOP:     MOVX    @R0,A
               INC     R0
               DJNZ    R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
               MOV     ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
               MOV     ?C_XBP,#HIGH XBPSTACKTOP
               MOV     ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
               MOV     ?C_PBP,#LOW PBPSTACKTOP
ENDIF

               MOV     SP,#?STACK-1
; This code is required if you use L51_BANK.A51 with Banking Mode 4
; EXTRN CODE (?B_SWITCH0)
;              CALL    ?B_SWITCH0 ; init bank mechanism to code bank 0
               LJMP    ?C_START
               END
```

SanctOS_V01.C

```
/*
***************************************************************************
*          SanctOS_V01.C                     RTOS KERNEL SOURCE CODE
*
*              Round Robin RTOS written in C by Ing. Paul P. Debono
*
* -------------------------------------------------------------------------
*
* For use with the 8051 family of microcontrollers
*
* Notes:
*
* Use NOOVERLAY in the linker BL51 Misc (Misc controls) options tab
* Use NOAREGS in the compiler C51 (Misc controls) options tab
*
* Timer to use for the RTOS ticks is user selectable, Timer 0, 1 or 2
* Naturally, Timer 2 can only be used with an 8032 CPU type.
*            Timer 1 can only be used if it is not required for
*            Baudrate generation
*
* Assign the correct values to
* 'STACKSIZE', 'TICK_TIMER', 'TICKTIME', 'CPU' and 'NOOFTASKS'
* in parameters.h
* Most of the time you need only to change 'NOOFTASKS' to reflect
* application
*
```

```
* If it is noticed that timing parameters are not being met,the system's
* TICKTIME can be modified by changing the value 'TICKTIME' in
* Parameters.H
* Please adhere to the conditions mentioned in Parameters.H
*
* File         : SanctOS_V01.C
* Revision     : 8
* Date         : February 2006
* By           : Paul P. Debono
*
*                      B. Eng. (Hons.) Elec. Course
*                      University Of Malta
*
***************************************************************************
*/

/*
***************************************************************************
*                                     INCLUDES
***************************************************************************
*/

#include <reg52.h>            /* 8052 Special Function Registers 8052 */
#include <SanctOS_V01.H>      /* RTOS system calls definitions */
                              /* (IN PROJECT DIRECTORY)*/
/*
***************************************************************************
*                              FUNCTION DEFINITIONS
***************************************************************************
*/

void PE_TaskChange (void);           /* used internally by the RTOS */

/*
***************************************************************************
*                      STATUS FLAG DEFINITIONS
/* if more flags are needed, use spare bits from status1 variable
***************************************************************************
*/
/* status1 - free bits for future expansion */
#define FLAG0_F        0x01    /* bit 0 - */
#define FLAG1_F        0x02    /* bit 1 - */
#define FLAG2_F        0x04    /* bit 2 - */
#define FLAG3_F        0x08    /* bit 3 - */
#define FLAG4_F        0x10    /* bit 4 - */
#define FLAG5_F        0x20    /* bit 5 - */
#define FLAG6_F        0x40    /* bit 6 - */
#define FLAG7_F        0x80    /* bit 7 - */

struct task_param xdata task[NOOFTASKS];

/*
***************************************************************************
*                              GLOBAL VARIABLES
***************************************************************************
*/
```

```
uchar data Running;            // Current task      number
uchar data tsknum;
/*
****************************************************************************
*/

/*
****************************************************************************
* RTOS FUNCTION DEFINITIONS
****************************************************************************
*/

/*
****************************************************************************
*
* Function name : OS_INIT_RTOS
*
* Function type : Initialisation System call
*
* Description   : This system call initialises the RTOS variables,
*                 task SPs and enables any required interrupts
*
* Arguments     : iemask Represents the interrupt enable mask which is
*                        used to set up the IE special function register.
*                     Its value determines which interrupts will be enabled
*                     during the execution of the user's application.
*
* Returns       : None
*
****************************************************************************
*/

void OS_INIT_RTOS(uchar iemask)
{
       uchar data i,j;

       #if (TICK_TIMER == 2)
         IE = (iemask & 0x7f) | 0x20;
 /* Set up 8051 IE register, using timer 2 */
         IP = 0x20;    /* Assign scheduler interrupt high priority */
       #elif (TICK_TIMER == 1)
         IE = (iemask & 0x7f) | 0x08;
/* Set up 8051 IE register, using timer 1 */
         IP = 0x08;    /* Assign scheduler interrupt high priority */
       #elif (TICK_TIMER == 0)
         IE = (iemask & 0x7f) | 0x02;
/* Set up 8051 IE register, using timer 0 */
         IP = 0x02;    /* Assign scheduler interrupt high priority */
       #endif

       Running = IDLE_TASK;          /* Set idle task, the running task, initially */
      tsknum = MINUS_ONE;
```

```
        for (i=0; i < NOOFTASKS; i++)
        {
           task[i].status1 = ZERO;            /* status flags (not used) */
           task[i].slot_time = TEN;           /* slot time counter value */
            task[i].slot_reload = TEN;        /* slot time reload value */
           task[i].stackptr = MAINSTACK + 6; /* SP storage */
                       /* clear bank 0 registers storage area */
           task[i].reg0 = ZERO;
           task[i].reg1 = ZERO;
           task[i].reg2 = ZERO;
           task[i].reg3 = ZERO;
           task[i].reg4 = ZERO;
           task[i].reg5 = ZERO;
           task[i].reg6 = ZERO;
           task[i].reg7 = ZERO;
                /* and clear the stack area */
           for (j=0;j<STACKSIZE;j++) task[i].stack[j]=ZERO;
        }
}

/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : OS_CREATE_TASK
*
* Function type : Initialisation System call
*
* Description   : This system call is used in the main program for each
*                              task to be created for use in the application.
*
* Arguments     : task_num Represents the task number
*                      (1st task is numbered as 0).
*
*                 task_add Represents the task's start address, which in
*                             the C environment, would simply be the name * * of the procedure
*                 slot_time Represents the number of ticks that this task
*                      will run before handing over to the next task
*
* Returns       : None
*
***************************************************************************
*/

void OS_CREATE_TASK(uchar task_num, uint task_add, uint slot)
{
        task[task_num].status1        = ZERO;   /* task flags not used */
          task[task_num].slot_time    = slot;        /* slot time value */
          task[task_num].slot_reload  = slot;
          task[task_num].stack[0]     = LoByte(task_add);    /* Little Endian */
        task[task_num].stack[1] = HiByte(task_add);          /* Low byte first */
}
```

```
/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : OS_RTOS_GO
*
* Function type : Initialisation System call
*
* Description   : This system calls is used to start the RTOS going such
*                     that it supervises the application processes.
*
* Arguments     : None
*
* Returns       : None
*
***************************************************************************/

void OS_RTOS_GO(void)
{
       #if (TICK_TIMER == 2)
       RCAP2H = HiByte(BASIC_TICK);  /* Configures Timer 2 in 16-bit       */
       RCAP2L = LoByte(BASIC_TICK);  /* auto-reload mode for the 8032      */
       T2CON = 0x84;          /* TR2 = TF2 = 1, causes immediate interrupt */

       #elif (TICK_TIMER == 0)
       TH0 = HiByte(BASIC_TICK);     /* Configure Timer 0 in 16-bit        */
       TL0 = LoByte(BASIC_TICK);     /* timer mode for the 8051    */
      TMOD &= 0xF0;    /* Clear T0 mode control, leaving T1 untouched */
       TMOD |= 0x01;  /* Set T0 mode control */
       TR0 = 1;       /* Start timer 0 */
       TF0 = 1;              /* Cause first interrupt immediately */

       #elif (TICK_TIMER == 1)

       TH1 = HiByte(BASIC_TICK);     /* Configure Timer 1 in 16-bit        */
       TL1 = LoByte(BASIC_TICK);     /* timer mode for the 8051    */
      TMOD &= 0x0F;    /* Clear T1 mode control, leaving T0 untouched */
       TMOD |= 0x10;  /* Set T1 mode control */
       TR1 = 1;          /* Start timer 1 */
       TF1 = 1;               /* Cause first interrupt immediately */

       #endif

       EA = 1;
/* Interrupts are enabled, starting the RTOS at this point. */
}

/*
***************************************************************************/

/*
***************************************************************************/
```

```
/*
****************************************************************************
* Function name : PE_TaskChange
*
* Function type : Context Switch (Internal function)
*
* Description   : This function is used to perform a forced or pre-emptive
*                 context switch or task swap
*
* Arguments     : none
*
*
* Notes         : This procedure is called from the timer tick interrupt,
*                 there would be 5 registers pushed on the stack, saved
*                 while the current task was running.
*                 Push A, B, DPH, DPL and PSW
*
*                 Comes here ONLY from an Interrupt Service Routine
*
* Returns       : None
*
****************************************************************************/

void PE_TaskChange (void) using 1
{
        uchar data i, temp;
        uchar idata * idata internal;
```

```
/* The current task is PRE-EMPTIED, and a */
/* new task is set to run */

/* NOW WORK WITH THE NEW TASK */

        internal = MAINSTACK;  /* MAINSTACK is the address of the start */
                               /* of main stack defined in STARTUP.A51 */
        tsknum++;       /* valid range 0 to (NOOFTASKS-1) */
        if (tsknum == NOOFTASKS) tsknum = ZERO;
        Running = tsknum;              /* set the new task as running */

    /* The new running task's USED stack area is copied to internal RAM */
    /* and the stack pointer adjusted accordingly */

      temp = task[Running].stackptr;
      i=0;
      do {
                *(internal++) = task[Running].stack[i++];
         } while (internal<=temp);
         SP = temp;     /* The new running task's SP is restored */

        /* Get the new tasks bank 0 registers which were stored externally */
           RecallBank0(&task[Running].reg0);
           /* then pop the SFRs back again and start other task here */
           POP5I();

/* it never gets down to here */
}

/*
****************************************************************************/

/*
***************************************************************************
* Function name : RTOS_Timer_Int
*
* Function type : Scheduler Interrupt Service Routine
*
* Description   : This is the RTOS scheduler ISR.
*                 It generates system ticks
*                 and calculates any remaining
*                   running time for each task.
*
* Arguments     : None
*
* Returns       : None
*
****************************************************************************/

#if (TICK_TIMER == 0)
        /* If Timer 0 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 1 using 1
{
         uchar idata * idata internal;
         uchar data k;
```

```
/* After an interrupt, the SP is incremented by 5 by the */
/* compiler to PUSH ACC,B,DPH,DPL and PSW */
/* These are popped back before returning from the interrupt */

         TH0 = HiByte(BASIC_TICK);   /* Timer registers reloaded */
         TL0 = LoByte(BASIC_TICK);

#elif (TICK_TIMER == 1)       /* If Timer 1 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 3 using 1
{
         uchar idata * idata internal;
         uchar data k;

/* After an interrupt, the SP is incremented by 5 by the */
/* compiler to PUSH ACC, B, DPH, DPL and PSW */
/* These are popped back before returning from the interrupt */
/* PSW is also pushed because of the 'using 1' command */
         TH1 = HiByte(BASIC_TICK);   /* Timer registers reloaded */
         TL1 = LoByte(BASIC_TICK);

#elif (TICK_TIMER == 2)       /* If Timer 2 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 5 using 1
{
         uchar idata * idata internal;
         uchar data i,k;

/* After an interrupt, the address of the next instruction of the */
/* current task is push on stack (low then high byte). Then SP */
/* is further incremented by 5 by the */
/* compiler to PUSH ACC,B,DPH,DPL and PSW */

/* Internal stack map at this stage */
/*   High stack RAM           */
/*   PSW <-- SP points to here        */
/*   DPL                              */
/*   DPH                              */
/*   B                                */
/*   ACC                              */
/*   High byte return address         */
/*   Low byte return address          */
/*   Low stack RAM                    */

/* These are normally popped back BEFORE returning from the */
/* interrupt IF the TaskChange function is not called. */

        TF2 = 0;                     /* Timer 2 interrupt flag is cleared */

#endif

     EA = 0;

     if (Running != IDLE_TASK)
      {

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler after */
```

```
/* the interrupt */
/* and are saved as part of the task stack */

     SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

/* check if the currently running task slot time has elapsed */

                       task[Running].slot_time--;
                       if (task[Running].slot_time == ZERO)
                       {
     /* Current task SP is saved pointing to PSW which is the last one */
     /* pushed on stack after the interrupt */
               task[Running].stackptr = k = SP;
                  internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

               i = 0;
               do {     /* Current task's USED stack area is saved */
                        task[Running].stack[i++] = *(internal++);
                    } while (internal<=k);

               task[Running].slot_time = task[Running].slot_reload;
               PE_TaskChange();

/* Force a pre-emptive task change if required */
/* Note that the pushed registers would still be on the saved stack at */
/* this point and would be popped back when task is put into */
/* action again in PE_TaskChange() */
               }
        }

/* else if running IDLE (after INIT_RTOS), start a task immediately */
/* without any need to save the stack, since the IDLE TASK will never */
/* run again in this round robin rtos. */

   else if (Running == IDLE_TASK) PE_TaskChange();

/* exits here if slot time for current task not yet over */
   EA = 1;
}

/*
***************************************************************************
*/

/*
***************************************************************************
***************************************************************************
***************************************************************************
***************************************************************************
***************************************************************************
*/
```

Appendix D PaulOS.C

This is the program source listing for the C version of PaulOS RTOS. It consists of:

- The header file PaulOS_V14_Parameters.h
- The header file PaulOS_V14.h
- The startup file PaulOS_Startup.A51
- The main source program PaulOS.C

PaulOS_V14_Parameters.h

```
/*
************************************************************************
*              PaulOS_v14_Params.H -- RTOS USER DEFINITIONS
************************************************************************
*/

#ifndef __paulos_v14_params_h__
#define __paulos_v14_params_h__
/************************************************************************/
```

```
// Number of bytes to allocate for the stack
#define STACKSIZE       0x0F


// set to 8051 or 8032
#define CPU                     8032

// Set to 0, 1 or 2 to select which timer to use as the RTOS tick timer
#define TICK_TIMER      2

// Length of RTOS basic tick in msec - Refer to RTOS timing definitions
// Suitable values: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50
#define TICKTIME        1

// Number of tasks used in application
#define NOOFTASKS       2

// Interrupts - set to 1 to use interrupt as stand alone ISR
#define STAND_ALONE_ISR_00 0   // EXT0
#define STAND_ALONE_ISR_01 1   // TIM0
#define STAND_ALONE_ISR_02 0   // EXT1
#define STAND_ALONE_ISR_03 0   // TIM1
#define STAND_ALONE_ISR_04 0   // SER0
#define STAND_ALONE_ISR_05 0   // TIM2

/**************************************************************************/

#endif // __paulos_v14_params_h__
```

PaulOS_V14.h

```
/*
 ************************************************************************
 *
 *      PaulOS_V14.H
 *      RTOS KERNEL HEADER FILE
 *
 * For use with PaulOS_V14.C,
 * A Co-Operative RTOS written in C by Ing. Paul P. Debono
 *
 * -----------------------------------------------------------------------
 *
 * For use with the 8051 family of microcontrollers
 *
 * File         : PaulOS_V14.C
 * Revision     : 2
 * Date         : April 2009
 * By           : Paul P. Debono
 *                        B. Eng. (Hons.) Elec. Course
 *                        University Of Malta
 *
 *
 ************************************************************************
 */

/*
```

```
        MAKE SURE THAT YOU ARE USING THE CORRECT STARTUP.A51 FILE, WHICH
        SHOULD INCLUDE THE FOLLOWING MAINSTACK DEFINITION.
        ENSURE ALSO THAT YOU HAVE THE CORRECT CSEG SETTING
*/
/*
                              RSEG    ?STACK
MAINSTACK:      DS      STACKSIZE     ; defined in parameters.h
                              EXTRN CODE (?C_START)
                              PUBLIC ?C_STARTUP
                              PUBLIC MAINSTACK
*/
#ifndef __PAULOS_V14_H__
#define __PAULOS_V14_H__

/*
 ************************************************************************
 *                                                   DATA TYPE DEFINITIONS
 ************************************************************************
 */
typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned long ulong;
#include "PaulOS_V14_Params.h"       /* in project directory */
/*
 ************************************************************************
 *                                             FUNCTION PROTOTYPES
 ************************************************************************
 */

/*
 * The following RTOS system calls do not receive any parameters :
 * ----------------------------------------------------------------
 */
// Stops current task and passes control to the next task in queue
void OS_DEFER(void);
// Kills the currently running task
void OS_KILL_IT(void);
// Checks if running task's signal bit is set
bit OS_SCHECK(void);
// Waits for end of task's periodic interval
void OS_WAITP(void);
// Returns the number of the currently executing (running) task
uchar OS_RUNNING_TASK_ID(void);
/* The following commands are simply defined as MACROS below
OS_CPU_IDLE()          Set the microprocessor into a sleep
                                      mode (awake every interrupt)
OS_CPU_DOWN()          Switch off microprocessor, activate
                                      only by hardware reset
OS_PAUSE_RTOS()               Disable RTOS, for stand alone ISR
OS_RESUME_RTOS()       Re-enable RTOS, for stand alone ISR
*/
/*
```

```
 * The following RTOS system calls do receive parameters :
 * -------------------------------------------------------
 */
// Initialises all RTOS variables
void OS_INIT_RTOS(uchar iemask);
// Starts the RTOS running with priorities if required
void OS_RTOS_GO(bit prior);
// Signals a task
void OS_SIGNAL_TASK(uchar tasknum);
// Waits for an event (interrupt) to occur
void OS_WAITI(uchar intnum);
// Waits for a timeout period given by a defined number of ticks
void OS_WAITT(uint ticks);
// Waits for a signal to arrive within a given number of ticks
void OS_WAITS(uint ticks);
// Sets task to run periodically every given number of ticks
void OS_PERIODIC(uint ticks);
// Creates a task
void OS_CREATE_TASK(uchar tasknum, uint taskadd);
// Resumes a task which was previously KILLed
void OS_RESUME_TASK(uchar tasknum);
/* The following commands are simply defined as MACROS below
OS_WAITT_A(M,S,ms)     Absolute WAITT for minutes, seconds, msecs
OS_WAITS_A(M,S,ms)     Absolute WAITS for minutes, seconds, msecs
OS_PERIODIC_A(M,S,ms)  Absolute PERIODIC for minutes, seconds, msecs
*/
/*************************************************************************/
```

```
/* The stack variable points to the start pointer in hardware stack and */
/* should be defined in PaulOS_STARTUP.A51 */
extern idata unsigned char MAINSTACK[STACKSIZE];
/*
 ************************************************************************
 *                                            RTOS TIMING DEFINITIONS
 ************************************************************************
 */
// In theory 921.6 counts represent 1 ms using an 11.0592 MHz crystal.
// Hence 9216 counts represent 10 ms.
#define MSEC10          9216UL
/*
Note on TICKTIME:
 Ensure that TICKTIME's value is chosen such that this quotient
 and hence all the following quotients result in an integer.
 In theory, maximum value of TICKTIME is given by the value corresponding
 to CLOCK = 65535, i.e. approx. 70-72.
 However respecting the condition above, max acceptable TICKTIME is 50 ms.
 Hence all suitable values are: 1, 2, 4, 5, 8, 10, 20, 25, 40, 50
 For reliable time-dependent results a value of 10 or above is recommended
 depending upon the application
*/
#define TICKS_PER_SEC   (1000 / TICKTIME)
#define TICKS_PER_MIN   (60000 / TICKTIME)
#define CLOCK           ((TICKTIME * MSEC10)/10UL)
#define BASIC_TICK      (65536 - CLOCK)
//An indefinite period of waiting time in the RTOS is given by a value 0
#define NOT_TIMING             0
// Indicates task not waiting for an interrupt
#define NO_INTERRUPT    0xFF
/*
 ************************************************************************
 *                                                  RTOS MACROS
 ************************************************************************
 */
// Retrieve High / Low byte
#define HiByte(Num) (uchar)((uint)(##Num)>>8);
#define LoByte(Num) (uchar)((uint)(##Num)& 0x00FF);
// Sets the MCU in idle mode
#define OS_CPU_IDLE()          PCON |= 0x01
// Sets the MCU in power-down mode
#define OS_CPU_DOWN()          PCON |= 0x02
// Pause / Resume RTOS functions
#if (TICK_TIMER == 0)
        #define OS_PAUSE_RTOS() EA = ET0 = TR0 = 0
        #define OS_RESUME_RTOS() TR0 = ET0 = EA = 1
#elif (TICK_TIMER == 1)
        #define OS_PAUSE_RTOS() EA = ET1 = TR1 = 0
        #define OS_RESUME_RTOS() TR1 = ET1 = EA = 1
#elif (TICK_TIMER == 2)
        #define OS_PAUSE_RTOS() EA = ET2 = TR2 = 0
        #define OS_RESUME_RTOS() TR2 = ET2 = EA = 1
```

```
#endif
/*
 ***********************************************************************
 *                                            COMPILE-TIME ERROR TRAPPING
 ***********************************************************************
 */
#if (CPU != 8032) && (CPU != 8051)
        #error Invalid CPU Setting
#endif
#if (NOOFTASKS > 254)
        #error Number of tasks is greater than 254 tasks
#endif
#if 0
        #if (CPU == 8032)
                #if ((MAINSTACK + STACKSIZE) > 0x100)
                        #error Out of RAM Space. Please shift variables to XDATA
                #endif
        #elif (CPU == 8051)
                #if ((MAINSTACK + STACKSIZE) > 0x80)
                        #error Out of RAM Space. Please shift variables to XDATA
                #endif
        #endif
#endif
#if ((TICKTIME * 110592 / 120) > 65535)
        #error Tick time value > valid range of the timer counter setting
#endif
#if ((TICKTIME * 110592 / 120) < 65535) && ((1000 % TICKTIME) != 0)
#error Undesirable TICKTIME. Valid values 1,2,4,8,10,20,25,40 or 50 ms
#endif
#if (CLOCK > 65535)
        #error Timer > valid range. Please check TICKTIME and MSEC.
#endif
/*
 ***********************************************************************
 *                                             TASK-RELATED DEFINITIONS
 ***********************************************************************
 */
#define FLAG_SIG_RCVD   0x80 // Signal-received flag mask      1000 0000
#define FLAG_SIG_WAIT   0x40 // Waiting-for-signal flag mask 0100 0000
#define FLAG_PERIODIC   0x20 // Periodic Interval flag mask 0010 0000
/*
 Interrupt Number used for tasks waiting for an interrupt event
*/
#define EXT0_INT        0x00    // External 0 Interrupt number 0
#define TIM0_INT        0x01    // Timer 0    Interrupt number 1
#define EXT1_INT        0x02    // External 1 Interrupt number 2
#define TIM1_INT        0x03    // Timer 1    Interrupt number 3
#define SER0_INT        0x04    // UART 0      Interrupt number 4
#define TIM2_INT        0x05    // Timer 2    Interrupt number 5
// Main endless loop in application given a task number equal to NOOFTASKS
#define IDLE_TASK NOOFTASKS
/*
 ***********************************************************************
 *                              ENHANCED EVENT-WAITING ADD-ON MACROS
 ***********************************************************************
```

```
 *
 * These macros perform the same functions of WAITT, WAITS and PERIODIC
 * calls but rather than ticks they accept absolute time values as
 * parameters in terms of days, hours, minutes, seconds and millisecs.
 * This difference is denoted by the _A suffix - eg. WAITT_A() is the
 * absolute-time version of WAITT()
 *
 * Range of values accepted, (maximum 65535 TICKTIMES):
 *
 * Using a minimum TICKTIME of 1 msec :
 *             from 1 msecs to 1 min, 5 secs, 535 msecs
 *
 * Using a recommended TICKTIME of 10 msec :
 *             from 10 msecs to 10 mins, 55 secs, 350 msecs
 *
 * Using a maximum TICKTIME of 50 msec :
 *             from 50 msecs to 54 mins, 36 secs, 750 msecs
 *
 * If the conversion from absolute time to ticks results in 0 (all
 * parameters being 0 or overflow) this result is only accepted by
 * WAITS() by virtue of how the WAITT(), WAITS() and PERIODIC() calls were
 * written. In the case of the WAITT() and PERIODIC() calls the tick count
 * would automatically be changed to 1 meaning an interval of
 * eg. 50 msecs in case the TICKTIME is defined to be 50 msecs
 *
 * Liberal use of parentheses is made in the following macros in case the
 * arguments might be expressions
 *
 *************************************************************************
 */
#define TPM(M) (TICKS_PER_MIN*(##M))
#define TPS(S) (TICKS_PER_SEC*(##S))
#define TPMS(ms) ((##ms)/TICKTIME)
#define OS_WAITT_A(M,S,ms) OS_WAITT((uint)(TPM(M) + TPS(S) + TPMS(ms)))
#define OS_WAITS_A(M,S,ms) OS_WAITS((uint)(TPM(M) + TPS(S) + TPMS(ms)))
#define OS_PERIODIC_A(M,S,ms) OS_PERIODIC((uint)(TPM(M)+TPS(S)+TPMS(ms)))
/*
 *************************************************************************
 *                             Other functions used internally by the RTOS
 *************************************************************************
 */
// Task swapping function
void QShift(void);
// RTOS Scheduler ISR
void RTOS_Timer_Int(void);
// Function used by ISRs other than the RTOS Scheduler
void Xtra_Int(uchar task_intflag);
// External Interrupt 0 ISR
#if (!STAND_ALONE_ISR_00)
void Xtra_Int_0(void);
#endif
// Timer 0 ISR
#if ( (TICK_TIMER != 0 ) && (!STAND_ALONE_ISR_01) )
```

```
void Xtra_Int_1(void);
#endif
// External Interrupt 1 ISR
#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2(void);
#endif
// Timer 1 ISR
#if ( (TICK_TIMER != 1 ) && (!STAND_ALONE_ISR_03) )
void Xtra_Int_3(void);
#endif
// Serial Port ISR
#if (!STAND_ALONE_ISR_04)
void Xtra_Int_4(void);
#endif
// Interrupt 5 (Timer 2) - NOT AVAILABLE ON THE 8051
#if ( (TICK_TIMER != 2 ) && (!STAND_ALONE_ISR_05) )
void Xtra_Int_5(void);
#endif
/**************************************************************************/
#endif // __PAULOS_V14_H__
```

PaulOS_STARTUP.A51

```
$NOMOD51
;------------------------------------------------------------------------
; This file is part of the C51 Compiler package
; Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
;------------------------------------------------------------------------
; PaulOS_STARTUP.A51: This code is executed after processor reset.
;
; To translate this file use A51 with the following invocation:
;
; A51 PaulOS_STARTUP.A51
;
; To link the modified PaulOS_STARTUP.OBJ file to your application use
; the following
; BL51 invocation:
;
; BL51 <your object file list>, PaulOS_STARTUP.OBJ <controls>
;
;------------------------------------------------------------------------
;
; User-defined Power-On Initialization of Memory
;
; With the following EQU statements the initialization of memory
; at processor reset can be defined:
;
; the absolute start-address of IDATA memory is always 0
IDATALEN EQU 100H ; the length of IDATA memory in bytes for the 8032 (256 bytes).
;
XDATASTART EQU 0H ; the absolute start-address of XDATA memory
XDATALEN EQU 0H ; the length of XDATA memory in bytes.
;
PDATASTART EQU 0H ; the absolute start-address of PDATA memory
PDATALEN EQU 0H ; the length of PDATA memory in bytes.
;
;Notes: The IDATA space overlaps physically the DATA and BIT areas of the
; 8051 CPU. At minimum the memory space occupied from the C51
; run-time routines must be set to zero.
;------------------------------------------------------------------------
;
; Reentrant Stack Initilization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialise it:
;
; Stack Space for reentrant functions in the SMALL model.
IBPSTACK EQU 0 ; set to 1 if small reentrant is used.
IBPSTACKTOP EQU 0FFH+1 ; set top of stack to highest location+1.
;IBPSTACKTOP EQU 07FH+1 ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
XBPSTACK EQU 0 ; set to 1 if large reentrant is used.
XBPSTACKTOP EQU 0FFFFH+1; set top of stack to highest location+1.
;
```

```
; Stack Space for reentrant functions in the COMPACT model.
PBPSTACK EQU 0 ; set to 1 if compact reentrant is used.
PBPSTACKTOP EQU 0FFFFH+1; set top of stack to highest location+1.
;
;------------------------------------------------------------------------
;
; Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
; The following EQU statements define the xdata page used for pdata
; variables. The EQU PPAGE must conform with the PPAGE control used
; in the linker invocation.
;
PPAGEENABLE EQU 0 ; set to 1 if pdata object are used.
;
PPAGE EQU 0 ; define PPAGE number.
;
PPAGE_SFR DATA 0A0H ; SFR that supplies uppermost address byte
; (most 8051 variants use P2 as uppermost address byte)
;
;------------------------------------------------------------------------
; Standard SFR Symbols
ACC DATA 0E0H
B DATA 0F0H
SP DATA 81H
DPL DATA 82H
DPH DATA 83H
 NAME ?C_STARTUP
?C_C51STARTUP SEGMENT CODE
?STACK SEGMENT IDATA
#include "PaulOS_V14_Params.h"
 RSEG ?STACK
MAINSTACK: DS STACKSIZE
 EXTRN CODE (?C_START)
 PUBLIC ?C_STARTUP
 PUBLIC MAINSTACK
; FLT32 or MON51 should be define in A51 TAB in Target Options
$IF (MON51)
 CSEG AT 8000H ; FOR DEV BOARD MON-51 MONITOR PROG
$ELSEIF (FLT32)
 CSEG AT 8100H ; FOR FLT-32 DEV BOARD MONITOR PROG
$ELSE
 CSEG AT 0 ; FOR EEPROM
$ENDIF
?C_STARTUP: LJMP STARTUP1
 RSEG ?C_C51STARTUP
STARTUP1:
IF IDATALEN <> 0
 MOV R0,#IDATALEN - 1
 CLR A
IDATALOOP: MOV @R0,A
 DJNZ R0,IDATALOOP
ENDIF
IF XDATALEN <> 0
 MOV DPTR,#XDATASTART
```

```
 MOV R7,#LOW (XDATALEN)
 IF (LOW (XDATALEN)) <> 0
 MOV R6,#(HIGH (XDATALEN)) +1
 ELSE
 MOV R6,#HIGH (XDATALEN)
 ENDIF
 CLR A
XDATALOOP: MOVX @DPTR,A
 INC DPTR
 DJNZ R7,XDATALOOP
 DJNZ R6,XDATALOOP
ENDIF
IF PPAGEENABLE <> 0
 MOV PPAGE_SFR,#PPAGE
ENDIF
IF PDATALEN <> 0
 MOV R0,#LOW (PDATASTART)
 MOV R7,#LOW (PDATALEN)
 CLR A
PDATALOOP: MOVX @R0,A
 INC R0
 DJNZ R7,PDATALOOP
ENDIF
IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
 MOV ?C_IBP,#LOW IBPSTACKTOP
ENDIF
```

```
IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
 MOV ?C_XBP,#HIGH XBPSTACKTOP
 MOV ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF
IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
 MOV ?C_PBP,#LOW PBPSTACKTOP
ENDIF
 MOV SP,#?STACK-1
; This code is required if you use L51_BANK.A51 with Banking Mode 4
; EXTRN CODE (?B_SWITCH0)
; CALL ?B_SWITCH0 ; init bank mechanism to code bank 0
 LJMP ?C_START
 END
```

PaulOS_V14.c

```
========================================================================
/*
 ************************************************************************
 *
 *      PaulOS_V14.C
 *      RTOS KERNEL SOURCE CODE
 *
 *      Co-Operative RTOS written in C by Ing. Paul P. Debono
 *
 * ----------------------------------------------------------------------
 *
 * For use with the 8051 family of microcontrollers
 *
 * Notes:
 *
 * Timer to use for the RTOS ticks is user selectable, Timer 0, 1 or 2
 * Naturally, Timer 2 can only be used with an 8032 CPU type.
 *
 * Assign the correct values to 'TICK_TIMER', 'CPU', 'MAINSTACK'
 * and 'NOOFTASKS' in PaulOS_V14_parameters.h
 *
 * If it is noticed that timing parameters are not being met,
 * the system's TICKTIME can be modified by changing the value 'TICKTIME'
 * in PaulOS_V14_parameters.H - please adhere to the conditions mentioned.
 *
 * File        : PaulOS_V14.C
 * Revision    : 2
 * Date        : April 2009
 * By          : Paul P. Debono
 *                      B. Eng. (Hons.) Elec. Course
 *                      University Of Malta
 *
 *
 ************************************************************************
 */
/*
```

```
 ***********************************************************************
 *                                                            INCLUDES
 ***********************************************************************
 */
// 8052 Special Function Registers 8032
#include "reg52.h"
// RTOS system calls definitions (in project directory)
#include "PaulOS_V14.h"
/*
 ***********************************************************************
 *                                             STRUCTURE DEFINITIONS
 ***********************************************************************
 */

// Task Parameters
struct task_param {
        uchar stackptr;                       // Stack pointer
        uchar flags;                  // Flags
        uchar intnum;         // Interrupt number task is waiting for
        uint timeout;                 // Timeout task is waiting for
        uint interval_count;  // Interval counter value
        uint interval_reload; // Interval reload value
        char stack[STACKSIZE]; // Stack contents
};
// Create instance for each user task (and IDLE task)
struct task_param xdata task[NOOFTASKS + 1];
/*
 ***********************************************************************
 *                                                    GLOBAL VARIABLES
 ***********************************************************************
 */
// Flag - task waiting for interrupt was found
bit bdata IntFlag;
// Flag - task timed out and ready to be placed in Ready Queue
bit bdata TinQFlag;
// Flag - priority is enabled/disabled
bit bdata Priority;
// Address of last ready task (pointer)
uchar data * data ReadyQTop;
// Number of the current running task
uchar data Running;
// Queue stack for tasks ready to run
uchar data ReadyQ[NOOFTASKS + 2];
/*
 ***********************************************************************
 *                                              FUNCTION DEFINITIONS
 ***********************************************************************
 */
/*
 ***********************************************************************
 *
 * Function name : OS_INIT_RTOS
 *
 * Function type : Initialisation System call
 *
```

```
 * Description : This system call initialises the RTOS variables,
 *                             task SPs and enables any required interrupts
 *
 * Arguments    : iemask      Represents the interrupt enable mask which
 *                                    is used to set up the IE special function
 *                                    register. Its value determines which
 *                                    interrupts will be enabled during the
 *                                    execution of the user's application.
 *
 * Returns      : None
 *
 *************************************************************************
 */
void OS_INIT_RTOS(uchar iemask) {
       uchar i, j;
#if (TICK_TIMER == 0)
       IE = (iemask & 0x7f) | 0x02;  // Set up 8051 IE register, timer 0
       IP = 0x02;                            // Give scheduler high priority
 #message "Using Timer 0 for the PaulOS rtos tick timer"
#elif (TICK_TIMER == 1)
       IE = (iemask & 0x7f) | 0x08;  // Set up 8051 IE register, timer 1
       IP = 0x08;                            // Give scheduler high priority
 #message "Using Timer 1 for the PaulOS rtos tick timer"
#elif (TICK_TIMER == 2)
       IE = (iemask & 0x7f) | 0x20;  // Set up 8051 IE register, timer 2
       IP = 0x20;                            // Give scheduler high priority
 #message "Using Timer 2 for the PaulOS rtos tick timer"
```

```
#endif
        // Set idle task as running task
        Running = IDLE_TASK;
        // Initialize each task
        for (i = 0; i < NOOFTASKS; i++) {
                // Clear timing, interrupt, interval & reload variables
                task[i].timeout = NOT_TIMING;
                task[i].intnum = NO_INTERRUPT;
                task[i].interval_count = NOT_TIMING;
                task[i].interval_reload = NOT_TIMING;

                // Fill READY queue with the idle task
                ReadyQ[i] = IDLE_TASK;
        }
        // Fill READY queue with the idle task
        ReadyQ[NOOFTASKS] = IDLE_TASK;
        ReadyQ[NOOFTASKS + 1] = IDLE_TASK;
        // Pointer to last task made to point to base of the queue
        ReadyQTop = ReadyQ;
        // For each task
        for (i = 0; i < NOOFTASKS + 1; i++) {
                /*
                 Initialise task SP values
                 SP initially set to point to MAINSTACK - 1
                 2 locations used to push return address and another push to
                 store PSW (done automatically by KEIL) in Qshift since
 we have
                 the USING 1 keyword.
                 Hence stackptr made to point to SP + 3 = MAINSTACK + 2
                */
                task[i].stackptr = MAINSTACK + 2;
                // Initialise task status bytes
                task[i].flags = 0;

                // Clear stack contents
                for (j = 0; j < STACKSIZE; j++) {
                        task[i].stack[j] = 0;
                }
        }
}
/*
 ***************************************************************************
 *
 * Function name : OS_CREATE_TASK
 *
 * Function type : Initialisation System call
 *
 * Description : This system call is used in the main program for each
 *                              task to be created for use in the application.
 *
 * Arguments    : tasknum     Represents the task number
 *                                                      (1st task is numbered as 0).
 *
 *              taskadd Represents the task's start address, which in
```

```
 *                                    the C environment, would simply be the name
 *                                    of the procedure
 *
 * Returns      : None
 *
 ***********************************************************************
 */
void OS_CREATE_TASK(uchar tasknum, uint taskadd) {
        // Add task to next available position in the READY queue
        ReadyQTop++;
        *ReadyQTop = tasknum;
        // Store task address (Little ENDIAN) on stack, ready for RET inst.
        task[tasknum].stack[0] = LoByte(taskadd);
        task[tasknum].stack[1] = HiByte(taskadd);
}
/*
 ***********************************************************************
 *
 * Function name : OS_RTOS_GO
 *
 * Function type : Initialisation System call
 *
 * Description : This system calls is used to start the RTOS going such
 *                             that it supervises the application processes.
 *
 * Arguments : prior   Determines whether tasks ready to be executed
 *                                    are sorted prior to processing or not.
 *
 *                            If prior = 0 a FIFO queue function is implied.
 *                            If prior = 1 the queue is sorted by task
 *                            number in ascending order, as a higher
 *                            priority is associated with smaller task
 *                            number (task 0 would have the highest
 *                            priority), such that the first task in the
 *                            queue, which would eventually run, would be
 *                            the one with the smallest task number having
 *                            the highest priority.
 *
 * Returns      : None
 *
 ***********************************************************************
 */
void OS_RTOS_GO(bit prior) {
        // Checks if tasks priorities are to be enabled
        Priority = prior;
#if (TICK_TIMER == 2)
        // Configures Timer 2 in 16-bit auto-reload mode for the 8032
        RCAP2H = HiByte(BASIC_TICK);
        RCAP2L = LoByte(BASIC_TICK);
        T2CON = 0x84; // TR2 = TF2 = 1
#elif (TICK_TIMER == 0)
        // Configure Timer 0 in 16-bit timer mode for the 8051
        TH0 = HiByte(BASIC_TICK);
```

```
        TL0 = LoByte(BASIC_TICK);
        TMOD &= 0xF0;  // Clear T0 mode control, leaving T1 untouched
        TMOD |= 0x01;  // Set T0 mode control
        TR0 = 1;               // Start timer 0
        TF0 = 1;               // Cause first interrupt immediately
#elif (TICK_TIMER == 1)
        // Configure Timer 1 in 16-bit timer mode for the 8051
        TH1 = HiByte(BASIC_TICK);
        TL1 = LoByte(BASIC_TICK);
        TMOD &= 0x0F;  // Clear T1 mode control, leaving T0 untouched
        TMOD |= 0x10;  // Set T1 mode control
        TR1 = 1;               // Start timer 1
        TF1 = 1;               // Cause first interrupt immediately
#endif
        // Signals scheduler that tasks have been added to the queue
        TinQFlag = 1;

        // Interrupts are enabled, starting the RTOS
        EA = 1;
}
/*
 **************************************************************************
 *
 * Function name : OS_RUNNING_TASK_ID
 *
 * Function type : Inter-task Communication System call
 *
```

```
 * Description : This system call is used to check to get the number of
 *                            the current task.
 *
 * Arguments    : None
 *
 * Returns      : Number of currently running task from which it must be
 *                            called
 *
 ***********************************************************************
 */
uchar OS_RUNNING_TASK_ID(void) {
       return (Running);
}
/*
 ***********************************************************************
 *
 * Function name : OS_SCHECK
 *
 * Function type : Inter-task Communication System call
 *
 * Description : This system call is used to check if the current task
 *                    has its signal set. It tests whether there was any
 *              signal sent to it by some other task.
 *
 * Arguments    : None
 *
 * Returns      : bit 1 if its signal bit is set, 0 if not set
 *
 ***********************************************************************
 */
bit OS_SCHECK(void) {
       // Disable interrupts
       EA = 0;
       // If a signal is present, clear it and return 1
       if (task[Running].flags & FLAG_SIG_RCVD) {
              task[Running].flags &= ~FLAG_SIG_RCVD;
              EA = 1;
              return 1;
       }
       // If a signal is not present, return 0
       else {
              EA = 1;
              return 0;
       }
}
/*
 ***********************************************************************
 *
 * Function name : OS_SIGNAL_TASK
 *
 * Function type : Inter-task Communication System call
 *
 * Description : This system call is used to send a signal to another
 *                            task.
 *
```

```
 * Arguments     : tasknum     Represents the task to which a signal is
 *                                                   required to be sent.
 *
 * Returns       : None
 *
 *************************************************************************
 */
void OS_SIGNAL_TASK(uchar tasknum) {
        // Disable interrupts
        EA = 0;
        // If the task has been waiting for a signal
        if (task[tasknum].flags & FLAG_SIG_WAIT) {

                // Clear its signal sent/wait flags
                task[tasknum].flags &= ~FLAG_SIG_RCVD;
                task[tasknum].flags &= ~FLAG_SIG_WAIT;
                task[tasknum].timeout = NOT_TIMING;
                ReadyQTop++;
                *ReadyQTop = tasknum;
                TinQFlag = 1;
        }
        // If it was not waiting, then set its signal sent flag
        else {
                task[tasknum].flags |= FLAG_SIG_RCVD;
        }
```

```
        // Re-enable interrupts
        EA = 1;
}
/*
 **************************************************************************
 *
 * Function name : OS_WAITI
 *
 * Function type : Event-Waiting System call
 *
 * Description : This system call causes task to wait for a given event
 *                     (interrupt). It identifies which interrupt the task
 *                     has to wait for. Once identified - the task's
 *                     appropriate flag is set and the task is put in the
 *                     waiting state by causing a task swap - the task
 *                     would wait indefinitely for the interrupt if its
 *                     timeout variable would be set to 0 (NOT_TIMING).
 *
 * Arguments    : intnum      Represents the interrupt number associated
 *                                     with the given interrupt for which the
 *                                     calling task intends to wait
 *
 * Returns      : None
 *
 **************************************************************************
 */
void OS_WAITI(uchar intnum) {
        // Disable interrupts
        EA = 0;
        switch (intnum) {
#if (!STAND_ALONE_ISR_00)
                // Interrupt number 0
                case 0:
                        // Task made to wait for external interrupt 0
                        task[Running].intnum = EXT0_INT;
                        QShift();
                        break;
#endif
#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
                // Interrupt number 1
                case 1:
                        // Task made to wait for timer 0 interrupt
                        task[Running].intnum = TIM0_INT;
                        QShift();
                        break;
#endif
#if (!STAND_ALONE_ISR_02)
                // Interrupt number 2
                case 2:
                        // Task made to wait for external interrupt 1
                        task[Running].intnum = EXT1_INT;
                        QShift();
                        break;
```

```
#endif
#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
               // Interrupt number 3
               case 3:
                       // Task made to wait for timer 1 interrupt
                       task[Running].intnum = TIM1_INT;
                       QShift();
                       break;
#endif
#if (!STAND_ALONE_ISR_04)
               // Interrupt number 4
               case 4:
                       // Task made to wait for serial port interrupt
                       task[Running].intnum = SER0_INT;
                       QShift();
                       break;
#endif
#if ( (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )
               // Interrupt number 5
               case 5:
                       // Task made to wait for timer 2 interrupt
                       task[Running].intnum = TIM2_INT;
                       QShift();
                       break;
#endif
               // Default action, do nothing
               default:
                       EA = 1;
                       break;
       }
}
/*
 ***************************************************************************
 *
 * Function name : OS_WAITT
 *
 * Function type : Event-Waiting System call
 *
 * Description : This system call causes a task to go in the waiting
 *                              state for a timeout period given by a defined
 *                              number of RTOS ticks.
 *
 * Arguments    : ticks       Represents the number of ticks for which the
 *                                     task will wait. Valid range for this
 *                                     parameter is 1 to 65535.
 *                                     A zero waiting time parameter is set to 1 by
 *                                     the RTOS itself, since a zero would
 *                                     effectively kill the task, making it wait
 *                                     forever.
 *
 * Returns      : None
 *
 ***************************************************************************
 */
```

```
void OS_WAITT(uint ticks) {
        EA = 0;
        // Just a precaution
        if (ticks == 0) ticks = 1;
        // Task's timeout variable is updated and the task then enters the
        // waiting state.
        task[Running].timeout = ticks;
        QShift();
}
/*
 *************************************************************************
 *
 * Function name : OS_WAITS
 *
 * Function type : Event-Waiting System call
 *
 * Description : This system call causes a task to wait for a signal to
 *                      arrive within a given number of RTOS ticks.
 *                      If the signal is already present, the task continues
 *                      to execute.
 *
 * Arguments     : ticks       Represents the number of ticks for which the
 *                                     task will wait for a signal to arrive.
 *                                     Valid range for this argument is 0 to 65535.
 *                                     A value of 0 means waiting forever for a
 *                                     signal to arrive.
 *
```

```
 * Returns      : None
 *
 *************************************************************************
 */
void OS_WAITS(uint ticks) {
        // Disable interrupts
        EA = 0;
        // If signal already sent, clear the signal and continue to run task
        if (task[Running].flags & FLAG_SIG_RCVD) {
                task[Running].flags &= ~FLAG_SIG_RCVD;
                EA = 1;
        }
        // If signal is not present send task to waiting state
        // by causing a task switch
        else {
                task[Running].flags |= FLAG_SIG_WAIT;
                task[Running].timeout = ticks;
                QShift();
        }
}
/*
 *************************************************************************
*/
/*
 *************************************************************************
 *
 * Function name : OS_WAITP
 *
 * Function type : Event-Waiting System call
 *
 * Description : This system call is used by a task to wait for the
 *                      end of its periodic interval. If the interval has
 *                      already passed, the task continues to execute.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 *************************************************************************
 */
void OS_WAITP(void) {
        // Disable interrupts
        EA = 0;
        // If the periodic interval time has elapsed, clear flag and
        // the task continues to execute
        if ((task[Running].flags & FLAG_PERIODIC) == FLAG_PERIODIC) {
                task[Running].flags &= ~FLAG_PERIODIC;
                EA = 1;
        }
        // Else put task into waiting state
        else {
                task[Running].flags |= FLAG_PERIODIC;
                QShift();
        }
}
```

```
/*
 *********************************************************************
 *
 * Function name : OS_PERIODIC
 *
 * Function type : Event-Waiting System call
 *
 * Description : This system call causes a task to repeat its function
 *                              every given number of RTOS ticks.
 *
 * Arguments : ticks   Represents the length of the periodic
 *                             interval in terms of RTOS ticks, after which
 *                             the task repeats itself. Valid range for this
 *                             parameter is from 1 to 65535.
 *
 * Returns      : None
 *
 *********************************************************************
 */
void OS_PERIODIC(uint ticks) {
       // Disable interrupts
       EA = 0;
       // Just a precaution
       if (ticks == 0) ticks = 1;
       // Initialise task's periodic interval count and reload vars
       task[Running].interval_count = ticks;
       task[Running].interval_reload = ticks;

       // Re-enable interrupts
       EA = 1;
}
/*
 *********************************************************************
 *
 * Function name : OS_DEFER
 *
 * Function type : Task Suspension System call
 *
 * Description : This system call is used to stop the current task in
 *                      order for the next task in the queue to execute.
 *                      In the meantime the current task is placed in the
 *                      waiting queue, just waiting for 2 ticks.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 *********************************************************************
 */
void OS_DEFER(void) {
       // Disable interrupts
       EA = 0;
```

```
// Make task wait for 2 ticks, thus giving up its time for other tasks
        task[Running].timeout = 2;
        QShift();
}
/*
 **************************************************************************
 *
 * Function name : OS_KILL_IT
 *
 * Function type : Task Suspension System call
 *
 * Description : This system call kills the current task, by putting it
 *                     permanently waiting, such that it never executes again.
 *                     It also clears any set waiting signals which the task
 *                     might have.
 *
 * Arguments     : None
 *
 * Returns       : None
 *
 **************************************************************************
 */
void OS_KILL_IT(void) {
        // Disable interrupts
        EA = 0;
        // Clear task's flags
        task[Running].flags = 0;
```

```
        // Set it to wait forever
        task[Running].timeout = NOT_TIMING;

        // Set periodic interval count to zero (note reload is left intact!)
        task[Running].interval_count = 0;

        // No longer wait for any interrupt event
        task[Running].intnum = NO_INTERRUPT;

        // Cause a task switch
        QShift();
}
/*
 ***********************************************************************
 *
 * Function name : OS_RESUME_TASK
 *
 * Function type : Inter-task Communication System call
 *
 * Description : This system call is used to resume another KILLed task.
 *
 * Arguments : tasknum Represents the task which is to be restarted.
 *
 * Returns      : None
 *
 ***********************************************************************
 */
void OS_RESUME_TASK(uchar tasknum) {
        // Disable interrupts
        EA = 0;
        // If task was periodic, resume periodic task
        if (task[tasknum].interval_reload != 0) {
                task[tasknum].interval_count = 1;
        }
        // Otherwise resume a normal waiting task after 1 tick
        else {
                task[tasknum].timeout = 1;
        }
        // Make RUNNING task wait for 2 ticks,
        // thus giving up its time for other tasks
        task[Running].timeout = 2;
        QShift();
}
/*
 ***********************************************************************
 *
 * Function name : QShift
 *
 * Function type : Context Switcher (Internal function)
 *
 * Description : This function is used to perform a context switch
 *                             i.e. voluntarily swaps task
 *
```

```
 * Arguments     : None
 *
 * Returns       : None
 *
 *************************************************************************
 */
void QShift(void) using 1 {
        // Variables used below
        uchar data i, temp;
        uchar data * idata internal;
        uchar data * idata qtask;
        uchar data * idata qptr;
        // Clear task in queue flag
        TinQFlag = 0;
        // Save SP of current task
        task[Running].stackptr = SP;
        temp = SP;
        // Save USED stack area of current task
        internal = MAINSTACK;
        i = 0;
        do {
                task[Running].stack[i++] = *(internal++);
        } while (internal <= temp);
        // Shift READY queue down by one position
        qtask = ReadyQ;
        qptr = ReadyQ + 1;
        while (qtask <= ReadyQTop) {
                *qtask++ = *qptr++;
        }
        ReadyQTop--; // Decrement pointer to last task in queue
// Ensure that this pointer is never below the start of the READY queue
        if (ReadyQTop < ReadyQ)
                ReadyQTop = ReadyQ;
        /*
         If task priorities are enabled, the queue is sorted such that the
         highest priority task becomes the running task, i.e. the one having
         the smallest task number.
        */
        if (Priority == 1) {
                // Scan just once through the list
                qptr = ReadyQTop;
                while (qptr > ReadyQ) {
                        qptr--;
                        if (*qptr > *(qptr + 1)) {
                                temp = *qptr;
                                *qptr = *(qptr + 1);
                                *(qptr + 1) = temp;
                        }
                }
        }

        // The first task in the READY queue becomes the new running task
        Running = ReadyQ[0];
```

```
        // The new running task's USED stack area is copied to internal RAM
        temp = task[Running].stackptr;
        internal = MAINSTACK;
        i = 0;
        do {
                *(internal++) = task[Running].stack[i++];
        } while (internal <= temp);
// Restore new running task's SP such that the new task will execute.
        SP = temp;

        // Re-enable interrupts
        EA = 1;
}
/*
 ************************************************************************
 *
 * Function name : RTOS_Timer_Int
 *
 * Function type : Scheduler Interrupt Service Routine ( Tick Timer )
 *
 * Description : This is the RTOS scheduler ISR. It generates system
 *                              ticks and calculates any remaining waiting
 *                              and periodic interval time for each task.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 ************************************************************************
 */
#if (TICK_TIMER == 0)
void RTOS_Timer_Int(void) interrupt 1 using 1 {
        uchar data k;
        uchar data * idata q;
        bit data On_Q;
        // Reload timer registers
        TH0 = HiByte(BASIC_TICK);
        TL0 = LoByte(BASIC_TICK);
#elif (TICK_TIMER == 1)
void RTOS_Timer_Int(void) interrupt 3 using 1 {
        uchar data k;
        uchar data * idata q;
        bit data On_Q;
        // Reload timer registers
        TH1 = HiByte(BASIC_TICK);
        TL1 = LoByte(BASIC_TICK);
#elif (TICK_TIMER == 2)
void RTOS_Timer_Int(void) interrupt 5 using 1 {
        uchar data k;
        uchar data * idata q;
        bit data On_Q;
```

```
        // Clear timer 2 interrupt flag
        TF2 = 0;
#endif
        // Loop over each task
        for (k = 0; k < NOOFTASKS; k++) {

                // Update the task's periodic intervals (if applicable)
                if (task[k].interval_count != NOT_TIMING) {
                        task[k].interval_count--;

                        // Has the periodic interval elapsed?
                        if (task[k].interval_count == NOT_TIMING) {
                                task[k].interval_count = task[k].interval_reload;

                // If the task has been waiting for the period to elapse,
                // place it in the READY queue (if not there already)
                if ((task[k].flags & FLAG_PERIODIC) == FLAG_PERIODIC) {
                                        task[k].flags &= ~FLAG_PERIODIC;
                                        q = ReadyQ;
                                        On_Q = 0;
                                        while (q <= ReadyQTop) {
                                                if (k == *q) {
                                                        On_Q = 1;
                                                        break;
                                                }
                                                q++;
                                        }
                                        if (On_Q == 0) {
                                                ReadyQTop++;
                                                *ReadyQTop = k;
                                                TinQFlag = 1;
                                        }
                                }

                                // If the task was not waiting for this event,
                                // do not place in the ready queue.
                                else {
                                        task[k].flags |= FLAG_PERIODIC;
                                }
                        }
                }
                // Update the task's timeout variables (if applicable)
                if (task[k].timeout != NOT_TIMING) {
                        task[k].timeout--;

                        // If timeout elapses,
                        // place task in READY queue
                        if (task[k].timeout == NOT_TIMING) {
                                ReadyQTop++;
                                *ReadyQTop = k;
                                TinQFlag = 1;
                                task[k].flags &= ~FLAG_SIG_WAIT;
                        }
                }
        }
```

```
        // If the idle task is running, and tasks are known to reside in the
        // queue, a task switch is purposely induced so these tasks can run.
        if ((TinQFlag == 1) && (Running == IDLE_TASK))
                QShift();
}
/*
 *************************************************************************
 *
 * Function name : Xtra_Int_0
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the external 0 interrupt ISR whose associated
 *                              interrupt number is 0.
 *
 * Arguments     : None
 *
 * Returns       : None
 *
 *************************************************************************
 */
#if (!STAND_ALONE_ISR_00)
void Xtra_Int_0(void) interrupt 0 using 1 {
        EA = 0;
        Xtra_Int(EXT0_INT); // Pass EXT0_INT for ident purposes
}
#endif
```

```
/*
 **********************************************************************
 *
 * Function name : Xtra_Int_1
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the Timer 0 ISR whose associated interrupt
 *                     number is 1. It is only enabled if the 8051 Timer 0
 *                     is not already being used as the RTOS scheduler.
 *
 *                     Timer 0 interrupt is usually used for RTOS on the
 *                     basic 8051.
 *
 *              For the FLT-32 8032 it can only be used with the modified
 *              version 2 monitor EPROM, or you are intending to write
 *              it on an EEPROM, since it is used for the single step
 *              in the old version monitor EPROM.
 *
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 **********************************************************************
 */
#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
void Xtra_Int_1(void) interrupt 1 using 1 {
       EA = 0;
       Xtra_Int(TIM0_INT); // Pass TIM0_INT for ident purposes
}
#endif
/*
 **********************************************************************
 *
 * Function name : Xtra_Int_2
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the external 1 interrupt ISR whose associated
 *                             interrupt number is 2.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 **********************************************************************
 */
#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2(void) interrupt 2 using 1 {
       EA = 0;
       Xtra_Int(EXT1_INT); // Pass EXT1_INT for ident purposes
}
```

```
#endif
/*
 ************************************************************************
 *
 * Function name : Xtra_Int_3
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the Timer 1 ISR whose associated interrupt
 *                             number is 3.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 ************************************************************************
 */
#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
void Xtra_Int_3(void) interrupt 3 using 1 {
       EA = 0;
       Xtra_Int(TIM1_INT); // Pass TIM1_INT for ident purposes
}
#endif
/*
 ************************************************************************
 *
 * Function name : Xtra_Int_4
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the serial port ISR whose associated interrupt
 *                             number is 4.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 ************************************************************************
 */
#if (!STAND_ALONE_ISR_04)
void Xtra_Int_4(void) interrupt 4 using 1 {
       EA = 0;
       Xtra_Int(SER0_INT); // Pass SER0_INT for ident purposes
}
#endif
/*
 ************************************************************************
 *
 * Function name : Xtra_Int_5
 *
 * Function type : Interrupt Service Routine
 *
 * Description : This is the Timer 2 ISR whose associated interrupt
```

```
 *                              number is 5.
 *
 * Arguments    : None
 *
 * Returns      : None
 *
 **************************************************************************
 */
#if ( (CPU == 8032) && (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )
void Xtra_Int_5(void) interrupt 5 using 1 {
       EA = 0;
       TF2 = 0;
       Xtra_Int(TIM2_INT); // Pass TIM2_INT for ident purposes
}
#endif
/*
 **************************************************************************
 *
 * Function name : Xtra_Int
 *
 * Function type : Interrupt Handling (Internal function)
 *
 * Description : This function performs the operations required by the
 *                              previous ISRs.
 *
 * Arguments : int_num Represents the flag mask for a given
 *                             interrupt against which the byte
 *                             storing the flags of each task will
 *                             be compared in order to determine
 *                             whether any task has been waiting
 *                             for the interrupt in question.
 *
 * Returns      : None
 *
 **************************************************************************
 */
void Xtra_Int(uchar int_num) using 1 {
       uchar data k;
       // To show if tasks have been affected by this interrupt
       IntFlag = 0;
       // For each task
       for (k = 0; k < NOOFTASKS; k++) {

              // If task has been waiting for the given interrupt
              if (task[k].intnum == int_num) {
                     // Clear the interrupt wait
                     task[k].intnum = NO_INTERRUPT;
                     IntFlag = 1;
                     task[k].timeout = NOT_TIMING;
                     ReadyQTop++;
                     *ReadyQTop = k;
              }
       }
```

```
        // If the IDLE task is running, and tasks are known to reside in the
        // READY queue, task switch is purposely induced so tasks can run.
        if ((IntFlag == 1) && (Running == IDLE_TASK)) {
                TinQFlag = 1;
                QShift();
        }
        // Otherwise if not IDLE task, the ISR exits after interrupts are
        // re-enabled, since RTOS cannot pre-empt task
        else if ((IntFlag == 1) && (Running != IDLE_TASK)) {
                TinQFlag = 1;
                EA = 1;
        }
        // Otherwise exit normally
        else EA = 1;
}
/*
*************************************************************************
*************************************************************************
*************************************************************************
*************************************************************************
*/
```

Appendix E MagnOS.C

This is the source listing of the RTOS program MagnOS (MAGNus Operating System, the latin word 'magnus' means great). It consists of:

- The assembly language 'include' file MagnOS_A01.A51
- The main program MagnOS.c
- The header file MagnOS.h
- The header file Parameters.h

MagnOS_A01.A51

```
; MagnOS_A01.A51
; MagnOS RTOS PLUS MAIN PROGRAM
;
; STORES ALL TASK REGISTERS
;
; Written by Paul P. Debono - JUNE 2006
; University of Malta
; Department of Communications and Computer Engineering
; MSIDA MSD 06; MALTA.
; Adapted and modified from the RTKB RTOS
; published in the book (CHAPTER 15)
; "C and the 8051 - 3rd Edition"
; by Thomas W. Schultz; Prentice Hall; ISBN 1-58961-237-X
;
; Accomodates many tasks, (take care of the stack size!)
;
; STACK MOVING VERSION - MOVES WORKING STACK IN AND OUT OF
; EXTERNAL MEMORY
; SLOWS DOWN RTOS, BUT DOES NOT RESTRICT TASK CALLS
; IDLE TASK (ENDLESS MAIN PROGRAM - TASK NUMBER = NOOFTASKS)
;
; THIS IS STILL A SMALL TEST VERSION RTOS. IT IS JUST USED FOR
; SHOWING WHAT IS NEEDED TO MAKE A SIMPLE RTOS.
; IT MIGHT STILL NEED SOME MORE FINE TUNING.
; IT HAS NOT BEEN THOROUGHLY TESTED !!!!
; WORKS FINE SO FAR.
; NO RESPONSABILITY IS TAKEN.

$NOMOD51
#include "reg52.h" ; check your own correct path
#include "Parameters.h"

/* The MAINSTACK variable points to the start pointer in hardware stack and */
/* is defined in STARTUP.A51 */
extrn idata (MAINSTACK)

PUBLIC _SaveBank0, _RecallBank0
PUBLIC _SaveSFRs, _RecallSFRs
PUBLIC POP5, POP0
PUBLIC POP5I, POP0I
```

```
; RTOS ASSEMBLY CODE MACROS
MAGNOS_DATA SEGMENT DATA
RSEG MAGNOS_DATA
TEMP1: DS 1
TEMP2: DS 1
TEMP3: DS 1
TEMP4: DS 1
TEMP5: DS 1

MAGNOS_ASM SEGMENT CODE
RSEG MAGNOS_ASM

POP5:
 DEC SP ; BLANK TO POP UNUSED RETURN ADDRESS
 DEC SP
 POP PSW
 POP DPL
 POP DPH
 POP B
 POP ACC
 SETB EA
 RET ; JUMPS TO PREVIOUSLY PRE-EMPTIED TASK HERE

POP0:
 DEC SP ; BLANK TO POP UNUSED RETURN ADDRESS
 DEC SP
 SETB EA
 RET ; JUMPS TO PREVIOUSLY PRE-EMPTIED TASK HERE

POP5I:
 DEC SP ; BLANK TO POP UNUSED RETURN ADDRESS
 DEC SP
 POP PSW
 POP DPL
 POP DPH
 POP B
 POP ACC
 SETB EA
 RETI ; JUMPS TO PREVIOUSLY PRE-EMPTIED TASK HERE

POP0I:
 DEC SP ; BLANK TO POP UNUSED RETURN ADDRESS
 DEC SP
 SETB EA
 RETI ; JUMPS TO PREVIOUSLY PRE-EMPTIED TASK HERE

_SaveSFRs:
 MOV TEMP1,ACC ; STORE A IN TEMP1
 MOV TEMP2,B ; STORE B IN TEMP2
 MOV TEMP3,DPH ; STORE DPH IN TEMP3
 MOV TEMP4,DPL ; STORE DPL IN TEMP4
 MOV ACC,PSW
 ANL A, #0E7H ; ensure stored PSW refers to bank 0 --> RS0=RS1=0 BANK 0
 MOV TEMP5,A ; STORE PSW in TEMP5
 MOV DPH,0EH ; R6 bank 1
```

```
MOV DPL,0FH ; R7 bank 1
MOV A,TEMP1
MOVX @DPTR,A ; SAVE ACC
INC DPTR
MOV A,TEMP2
MOVX @DPTR,A ; SAVE B
INC DPTR
MOV A,TEMP3
MOVX @DPTR,A ; SAVE DPH
INC DPTR
MOV A,TEMP4
MOVX @DPTR,A ; SAVE DPL
INC DPTR
MOV A,TEMP5
MOVX @DPTR,A ; SAVE PSW
RET

_RecallSFRs:
MOV DPH,0Eh ; get task store address, R6 bank 1
MOV DPL,0Fh
MOVX A,@DPTR ; GET ACC
MOV TEMP1,A ; STORE A IN TEMP1
INC DPTR
MOVX A,@DPTR ; GET B
MOV TEMP2,A ; STORE B IN TEMP2
INC DPTR
MOVX A,@DPTR ; GET DPH
MOV TEMP3,A ; STORE DPH IN TEMP3
```

```
 INC DPTR
 MOVX A,@DPTR ; GET DPL
 MOV TEMP4,A ; STORE DPL IN TEMP4
 INC DPTR
 MOVX A,@DPTR ; GET PSW
 MOV TEMP5,A ; STORE PSW IN TEMP 5
 MOV ACC,TEMP1 ; RESTORE A
 MOV B,TEMP2 ; RESTORE B
 MOV DPH,TEMP3 ; RESTORE DPH
 MOV DPL,TEMP4 ; RESTORE DPL
 MOV PSW,TEMP5 ; RESTORE PSW
 RET

_SaveBank0:             ; Address high byte in R6, low byte in R7 bank 1
 MOV DPH,0EH
 MOV DPL,0FH
 MOV A,0
 MOVX @DPTR,A
 INC DPTR
 MOV A,1
 MOVX @DPTR,A
 INC DPTR
 MOV A,2
 MOVX @DPTR,A
 INC DPTR
 MOV A,3
 MOVX @DPTR,A
 INC DPTR
 MOV A,4
 MOVX @DPTR,A
 INC DPTR
 MOV A,5
 MOVX @DPTR,A
 INC DPTR
 MOV A,6
 MOVX @DPTR,A
 INC DPTR
 MOV A,7
 MOVX @DPTR,A
 RET

_RecallBank0:               ; Address high byte in R6, low byte in R7 bank 1
 MOV DPH,0EH
 MOV DPL,0FH
 MOVX A,@DPTR
 MOV 0,A
 INC DPTR
 MOVX A,@DPTR
 MOV 1,A
 INC DPTR
 MOVX A,@DPTR
 MOV 2,A
 INC DPTR
 MOVX A,@DPTR
```

```
 MOV 3,A
 INC DPTR
 MOVX A,@DPTR
 MOV 4,A
 INC DPTR
 MOVX A,@DPTR
 MOV 5,A
 INC DPTR
 MOVX A,@DPTR
 MOV 6,A
 INC DPTR
 MOVX A,@DPTR
 MOV 7,A
 RET

END
```

MagnOS_V01.C

```
/*
**************************************************************************
*       MagnOS_V01.C RTOS KERNEL SOURCE CODE
*
* Pre-Emptive RTOS written in C by Ing. Paul P. Debono
*
* ------------------------------------------------------------------------
*
* For use with the 8051 family of microcontrollers
*
* Notes:
*
* Use NOOVERLAY in the linker BL51 Misc (Misc controls) options tab
* Use NOAREGS in the compiler C51 (Misc controls) options tab
*
* Timer to use for the RTOS ticks is user selectable, Timer 0, 1 or 2
* Naturally,   Timer 2 can only be used with an 8032 CPU type.
*              Timer 1 can only be used if it is not required for
*              Baudrate generation
*
* Assign the correct values to
* 'STACKSIZE', 'TICK_TIMER', 'TICKTIME', 'CPU' and 'NOOFTASKS'
* in parameters.h
* Most of the time you need only to change 'NOOFTASKS' to reflect the
* application
*
* If it is noticed that timing parameters are not being met,the system's
* TICKTIME can be modified by changing the value 'TICKTIME' in Parameters.H
* Please adhere to the conditions mentioned in Parameters.H
*
* File : MagnOS_V01.C
* Revision : 8
* Date : February 2006
* By     : Paul P. Debono
*
*                       B. Eng. (Hons.) Elec. Course
*                       University Of Malta
*
**************************************************************************
*/

/*
**************************************************************************
*                                    INCLUDES
**************************************************************************
*/

#include <reg52.h>            /* 8052 Special Function Registers 8052 */
#include <MagnOS_V01.H>       /* RTOS system calls definitions */
                              /* (IN PROJECT DIRECTORY)*/

/*
```

```
**************************************************************************
*                              FUNCTION DEFINITIONS
**************************************************************************
*/

void V_TaskChange (void); /* used internally by the RTOS */
void PE_TaskChange (void); /* used internally by the RTOS */

/*
**************************************************************************
*               STATUS FLAG DEFINITIONS
/* if more flags are needed, use spare bits from status1 or status2
* variable
**************************************************************************
*/
/* status1 - has some free bits for future expansion */

#define WAIT4I_F        0x01 /* bit 00 - task waiting for an interrupt */
#define WAIT4V_F        0x02 /* bit 01 - task waiting periodic interval */
#define WAIT4S_F        0x04 /* bit 02 - task waiting for a semaphore */
#define WAIT4M_F        0x08 /* bit 03 - task waiting for a message */

/* status2 - has some free bits for future expansion */

#define PREEMP_F        0x01 /* bit 00 - task was pre-emptied */
#define FIRST_TIME_F    0x02 /* bit 01 - task running the first time */
#define TASK_KILLED_F   0x04 /* bit 02 - task killed */
#define NO_MBOX_FREE_F  0x08 /* bit 03 - no mailbox space */

struct task_param xdata task[NOOFTASKS + 1];

struct letter xdata mbox[MBXSIZE];
/* MBXSIZE messages: destination, source, length */
/*                      plus 16 data characters per message */

/*
**************************************************************************
*                              GLOBAL VARIABLES
**************************************************************************
*/

bit bdata TinQFlag;
/* Flag indicating new higher priority task timed out and */
                      /* put in ReadyQ */

uchar data Resource[8];              // 8 resources available
uchar data * data ReadyQTop;         // Address of last ready task (point)
uchar data Running;                  // Current task number
uchar data ReadyQ[NOOFTASKS + 3];    // Queue stack for tasks ready to run

/*
**************************************************************************
*/

/*
**************************************************************************
* RTOS FUNCTION DEFINITIONS
**************************************************************************
*/
```

```
/*
***************************************************************************
*
* Function name : OS_INIT_RTOS
*
* Function type : Initialisation System call
*
* Description  : This system call initialises the RTOS variables, task
*                      SPs and enables any required interrupts
*
* Arguments    : iemask Represents the interrupt enable mask which is
*                      used to set up the IE special function register.
*                      Its value determines which interrupts will be
*                      enabled during the execution of the user's
*                      application.
*
* Returns      : None
*
***************************************************************************
*/

void OS_INIT_RTOS(uchar iemask)
{
       uchar data i,j;

       #if (TICK_TIMER == 2)
        IE = (iemask & 0x7f) | 0x20;
/* Set up 8051 IE register, using timer 2 */
```

```
        IP = 0x20;      /* Assign scheduler interrupt high priority */
         #elif (TICK_TIMER == 1)
           IE = (iemask & 0x7f) | 0x08;
/* Set up 8051 IE register, using timer 1 */
         IP = 0x08;     /* Assign scheduler interrupt high priority */
        #elif (TICK_TIMER == 0)
         IE = (iemask & 0x7f) | 0x02;
/* Set up 8051 IE register, using timer 0 */
         IP = 0x02;     /* Assign scheduler interrupt high priority */
        #endif

        Running = IDLE_TASK;    /* Set idle task as the running task */

        for (i=0; i <= NOOFTASKS; i++)
        {
                task[i].catalog = i;          /* task id */
                task[i].status1 = ZERO;
/* status flags, see below for details */
        task[i].status2 = ZERO;
/* status flags, see below for details */
        task[i].priority = LOWEST; /* priority flag */
        task[i].semaphore = ZERO;
/* counting semaphore for each task */
        task[i].resource = FREE;
/* resources requested for each task */
        task[i].stackptr = MAINSTACK + 1; /* SP storage */
        task[i].intnum = NO_INTERRUPT;
/* task not waiting for any interrupt */
        task[i].timeout = NOT_TIMING;
/* task not waiting for any timeout */
                task[i].interval_count = ZERO; /* task not periodic */
                task[i].interval_reload = ZERO;
/* periodic interval reload value */
/* clear registers storage area */

        task[i].rega = ZERO;
        task[i].regb = ZERO;
        task[i].rdph = ZERO;
         task[i].rdpl = ZERO;
        task[i].rpsw = ZERO;
        task[i].reg0 = ZERO;
        task[i].reg1 = ZERO;
        task[i].reg2 = ZERO;
        task[i].reg3 = ZERO;
        task[i].reg4 = ZERO;
        task[i].reg5 = ZERO;
        task[i].reg6 = ZERO;
        task[i].reg7 = ZERO;

/* clear stack storage area */
        for (j=0;j<STACKSIZE;j++) task[i].stack[j]=ZERO;

                ReadyQ[i] = IDLE_TASK;  /* Fill the READY queue */
        }                                   /* with the idle task */
```

```
        ReadyQ[NOOFTASKS + 1] = IDLE_TASK;
        ReadyQ[NOOFTASKS + 2] = IDLE_TASK;

        ReadyQTop = ReadyQ;     /* Pointer to last task made to point to */
                                /* base of the queue. */
/* Now clear mailboxes */

 for (i=0;i<MBXSIZE;i++)
  {
        mbox[i].dest = FREE;
        mbox[i].src = FREE;
        mbox[i].len = ZERO;
          for(j=0;j<DATASIZE;j++)
          mbox[i].dat.string.s[j] = ZERO;
        }
/* Now clear resources */

 for (i=0;i<NOOFRESOURCES;i++)
   {
        Resource[i] = FREE;
         }

}

/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : OS_CREATE_TASK
*
* Function type : Initialisation System call
*
* Description : This system call is used in the main program for each
*                      task to be
*                              created for use in the application.
*
* Arguments :  task_num Represents the task number
*              (1st task is numbered as 0).
*
*              task_add       Represents the task's start address,
*                             which in the C
*                                    environment, would simply be the name of the
*                             procedure
*               task_priority Represents the priority of the task
*                      0 is low priority, 255 is the highest (top) priority
*
* Returns : None
*
**************************************************************************
*/

void OS_CREATE_TASK(uchar task_num, uint task_add, uchar task_priority)
{
```

```
        ReadyQTop++; /* Increment queue pointer. Task is added to next */
        *ReadyQTop = task_num;/* available position in the READY queue.*/

        task[task_num].stack[0] = LoByte(task_add); /* Little Endian */
        task[task_num].stack[1] = HiByte(task_add); /* Low byte first */
        task[task_num].priority = task_priority;
        task[task_num].catalog = task_num;
        task[task_num].status2 |= FIRST_TIME_F;
/* task running for 1st time */

}

/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_RTOS_GO
*
* Function type : Initialisation System call
*
* Description   : This system calls is used to start the RTOS going such
*                     that it
*                              supervises the application processes.
*
* Arguments     : None
*
* Returns       : None
*
**************************************************************************
*/

void OS_RTOS_GO(void)
{
        #if (TICK_TIMER == 2)
        RCAP2H = HiByte(BASIC_TICK);  /* Configures Timer 2 in 16-bit       */
        RCAP2L = LoByte(BASIC_TICK);  /* auto-reload mode for the 8032      */
        T2CON = 0x84; /* TR2 = TF2 = 1 */

        #elif (TICK_TIMER == 0)

        TH0 = HiByte(BASIC_TICK);     /* Configure Timer 0 in 16-bit        */
        TL0 = LoByte(BASIC_TICK);     /* timer mode for the 8051    */
 TMOD &= 0xF0;         /* Clear T0 mode control, leaving T1 untouched */
        TMOD |= 0x01;          /* Set T0 mode control */
        TR0 = 1;       /* Start timer 0 */
        TF0 = 1;                      /* Cause first interrupt immediately */

        #elif (TICK_TIMER == 1)

        TH1 = HiByte(BASIC_TICK);     /* Configure Timer 1 in 16-bit        */
        TL1 = LoByte(BASIC_TICK);     /* timer mode for the 8051    */
 TMOD &= 0x0F; /* Clear T1 mode control, leaving T0 untouched */
        TMOD |= 0x10;          /* Set T1 mode control */
        TR1 = 1;               /* Start timer 1 */
```

```
        TF1 = 1;                      /* Cause first interrupt immediately */

        #endif

        TinQFlag = 1; /* Signals scheduler that tasks have been     */
                /* added to the READY queue, so that they may start to run. */
        EA = 1;
/* Interrupts are enabled, starting the RTOS at this point. */
}

/*
**************************************************************************
*/

/*
**************************************************************************
* Function name : OS_CHECK_TASK_PRIORITY
*
* Function type : Inter-task Communication System call
*
* Description : This system call is used to get the priority of the
*                               requested task.
*
* Arguments : None
*
* Returns : Relevant task priority
*
**************************************************************************
*/
```

```
uchar OS_CHECK_TASK_PRIORITY(uchar task_num)
{
        return task[task_num].priority;
}
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name       : OS_CHANGE_TASK_PRIORITY
*
* Function type       : Inter-task Communication System call
*
* Description         : This system call is used to change the priority of the
*                             requested task.
*
* Arguments           : None
*
* Returns             : None
*
**************************************************************************
*/
void OS_CHANGE_TASK_PRIORITY(uchar task_num, uchar new_prio)
{
        EA=0;
        task[task_num].priority = new_prio;
        EA=1;
}
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_RUNNING_TASK_ID
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to check to get the number of
*                       the current task.
*
* Arguments     : None
*
* Returns       : Number of currently running task from which it must
*                       be called
*
**************************************************************************
*/
uchar OS_RUNNING_TASK_ID(void)
{
        return (Running);
}
```

```
/*
************************************************************************
*/

/*
************************************************************************
*
* Function name : OS_CHECK_TASK_SEMA4
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to get the semaphore of the
*                              requested task.
*
* Arguments     : None
*
* Returns       : Relevant task semaphore
*
************************************************************************
*/
uchar OS_CHECK_TASK_SEMA4(uchar task_num)
{
       return task[task_num].semaphore;
}
/*
************************************************************************
*/

/*
************************************************************************
*
* Function name : OS_SEMA4_PLUS
*
* Function type : Add Units to a Semaphore System call
*
* Description   : This system adds units to a semaphore of a particular
*                      task
*                      No task change is involved
*
* Arguments     : task_num           Represents the task number
*                 units       Number of units to add to semaphore
*
* Returns       : None
*
***************************************************************************/

void OS_SEMA4PLUS (uchar task_num, uchar units)
{
 EA = 0;
 if (units > (255-task[task_num].semaphore))
       task[task_num].semaphore = MAXSEM;
 else task[task_num].semaphore += units;
 EA = 1;

}
```

```
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_SEMA4_MINUS
*
* Function type : Subtracts Units to a Semaphore System call
*
* Description   : This system subtracts units from semaphore of particular
*                 task
*                 If semaphore reaches ZERO, a voluntary task switch is
*                 invoked
*
* Arguments    :       task_num       Represents the task number
*                      units          Number of units to add to semaphore
*
* Returns : None
*
*************************************************************************
*/

void OS_SEMA4MINUS (uchar task_num, uchar units)
{
  uchar data i, temp;
  uchar idata * idata internal;
```

```
 EA = 0;

/*      store current task registers just in case task change is required */
        RS0 = 1; /* USE BANK 1 */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* RETURN TO BANK 0 */

                task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
                internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */
                /* Current task's USED stack area is saved */
 i = 0;
 do {
                task[Running].stack[i++] = *(internal++);
                } while (internal<=temp);

 if (units > task[task_num].semaphore) task[task_num].semaphore = ZERO;
 else task[task_num].semaphore -= units;

 if ((task[task_num].semaphore==ZERO)&&(task[task_num].status1&WAIT4S_F))
 {
                task[task_num].status1 &= ~WAIT4S_F;   /* clear flag  */
        task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */
        task[Running].timeout = 5;

/* Either use */
        task[task_num].timeout = 1;

/* or use */
//       task[task_num].timeout = NOT_TIMING;  /* clear flag  */
//       ReadyQTop++;
//       *ReadyQTop = task_num;
/* place the task which had been waiting for */
/* the semaphore in the ReadyQ */

         V_TaskChange();
        }
 EA = 1;
}
/*
*************************************************************************
*/

/*
*************************************************************************
*
* Function name :      OS_WAIT4SEM
*
* Function type :      Event-Waiting System call
*
* Description   :      This system call causes a task to wait for a semaphore to
*                      reach zero (within a given timeout), calling a voluntary
*                      task change.
*                      0 timeout implies wait forever. If the semaphore is
*                              already zero, the task continues to execute.
```

```
*
* Arguments : ticks           Represents the number of ticks for which the
*                       task will wait for the semaphore. Valid range for this
*                       argument is 0 to 65535. A value of 0 means waiting
*                       forever for the semaphore.
*
* Returns      :       None
*
**************************************************************************
*/

void OS_WAIT4SEM (uint ticks)
{
       uchar data i, temp;
               uchar idata * idata internal;

       EA = 0;

/* store current task registers just in case task change is required */
       RS0 = 1; /* use bank 1 */

/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
       SaveSFRs(&task[Running].rega);
       SaveBank0(&task[Running].reg0);
       RS0 = 0; /* use bank 0 */

        task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
        internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {             /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
               } while (internal<=temp);

       if (!task[Running].semaphore)         /* If semaphore already */
       {                                             /* zero it clears the */
               task[Running].status1 &= ~WAIT4S_F;    /* flag and the task */

       EA = 1;                          /* continues to run. */
       }
       else
       {                                      /* If semaphore is not present */
               task[Running].status1 |= WAIT4S_F; /* task is sent in the */
               task[Running].timeout = ticks; /* waiting state, setting */
       task[Running].status2 &= ~PREEMP_F; /* mark task NOT pre-emptied */

       V_TaskChange();                        /* a task switch.      */
       }
}
/*
**************************************************************************
*/

/*
**************************************************************************
*
```

```
* Function name : OS_RELEASE_RES
*
* Function type : Releases a resource System call
*
* Description   : This system releases a resource for other tasks to use
*                  If there are other tasks waiting, the task with the
*                  highest priority is made ready to execute and a voluntary
*                  task switch is invoked
*
* Arguments     : res_num            Represents the resource number
*
* Returns : None
*
**************************************************************************
*/

void OS_RELEASE_RES (uchar Res_Num)
{
  uchar data i, temp, tp;
  uchar idata * idata internal;
  bit found;

  EA = 0;

/* store current task registers just in case task change is required */
        RS0 = 1; /* USE BANK 1 */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* RETURN TO BANK 0 */

                task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
                internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */
                /* Current task's USED stack area is saved */
        i = 0;
        do {
                        task[Running].stack[i++] = *(internal++);
                        } while (internal<=temp);
        tp=0;
        found=0;
/* first find highest priority task that was waiting for this resource */
/* or variable to be free */
 for(i=0;i<NOOFTASKS;i++)
    {
    if ((task[i].resource==Res_Num) && task[i].priority>tp)
        { temp=i;
          tp= task[i].priority;
          found=1;
        }
 }

 if(found)      /* temp now contains task number of the highest priority */
                /* task that was waiting for the resource */
        {
        Resource[Res_Num] = temp;/* mark resource as being used by new task */
//       task[temp].status1 &= ~WAIT4R_F;      /* clear flag  */
        task[temp].resource = FREE;
```

```
/* mark task no longer waiting for resource */
       task[Running].timeout = 3; /* put running task as waiting timeout */
       task[Running].status2 &= ~PREEMP_F; /* mark task as NOT pre-emptied */

/* Either use */
//      task[temp].timeout = 1;
/* or use */
        task[temp].timeout = NOT_TIMING;      /* clear flag  */
        ReadyQTop++;
        *ReadyQTop = temp;    /* place the task which had been waiting for */
                              /* the semaphore in the ReadyQ */

        V_TaskChange();
 }

 else
 {
       Resource[Res_Num] = FREE;
       EA = 1;
 }
}
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_WAIT4RES
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to wait for a resource to
*                 become zero (within a given timeout),
*                 calling a voluntary task
*                 change. 0 timeout implies wait forever.
*                     If the resource is
*                             already available, the task continues to execute.
*
* Arguments     : ticks               Represents the number of ticks for which the
*                             task will wait for the semaphore. Valid range for
*                                   this argument is 0 to 65535. A value of 0 means
*                            waiting forever for the semaphore.
*
* Returns       : None
*
**************************************************************************
*/

void OS_WAIT4RES (uchar Res_Num,uint ticks)
{
 uchar data i, temp;
       uchar idata * idata internal;

       EA = 0;
```

```
/* store current task registers just in case task change is required */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */
 i = 0;
 do {                  /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=temp);

        if (Resource[Res_Num]==FREE) /* If resource already */
        {                                              /* available it takes the */
                Resource[Res_Num]=Running;     /* resource and the task */
        task[Running].resource=FREE;
        EA = 1;                                        /* continues to run. */
        }
        else
        {                                      /* If resource is being used */
//              task[Running].status1 |= WAIT4R_F;
/* the task is sent in the */
                task[Running].resource = Res_Num;
/* waiting state, task change */
                task[Running].timeout = ticks;
 task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */
        V_TaskChange();                        /* a task switch.     */
        }
}

/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_SEND_MSG
*
* Function type : task suspension system call
*
* Description   : this system call sends a message to another task
*                 If other task was already waiting, a voluntary task
*                             change is invoked.
*
* Arguments    :   message, with the following structure
*                 struct letter{uchar dest,src;union dataformat dat;}
* Returns      :   none
*
**************************************************************************/
```

```
void OS_SEND_MSG(struct letter xdata *msg)
{
 uchar i,j,temp,dest_task,msg_len;
 uchar idata * idata internal;
 bit waiting,mboxfree;

 EA=0;

 RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there isa need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

        task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
        internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                  /* Current task's USED stack area is saved */
               task[Running].stack[i++] = *(internal++);
               } while (internal<=temp);
 i=0;
 waiting=0;
 dest_task = msg[0].dest;
 msg_len = msg[0].len;

        do {
               if((mbox[i].dest==dest_task) && (mbox[i].src==FREE) && (mbox[i].len==ZERO) &&
```

```
(task[dest_task].status1 & WAIT4M_F))
 /* there is already task waiting for message */
 /* Hence transfer message, clear mailbox and make a
    voluntary task change */
               {
       waiting=1;
       mbox[i].src = msg[0].src;
       mbox[i].len = msg_len;
       for (j=0;j<msg_len;j++)
       mbox[i].dat.string.s[j] = msg[0].dat.string.s[j];
/* Place the task that was waiting for message, in Ready Q */
/* Leave the WAIT4M_F still set, since it will be used */
/* and cleared later on */
/* by the GET_MSG routine */

/* Either use */
 task[dest_task].timeout = NOT_TIMING;
                /* mark task as NOT waiting for timeout */
                ReadyQTop++;
                *ReadyQTop = dest_task;

/* or use */
//       task[dest_task].timeout = 1;
                /* mark task as waiting for 1 timeout */

/* and then the task change is made here */
/* where the running task enters a waiting state */
/* The mailbox has to be cleared by issuing the 'clear message command' */
/* immediately after using the 'wait message' command */
/* This is done automatically when you use the 'wait4msg' command */

               task[Running].timeout = 2;
       task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-empted */
        V_TaskChange();              /* a task switch.      */
               }
       i++;
               } while((i<MBXSIZE) && !waiting);
/* Else, find free mailbox location
/* and leave message */
 if(!waiting)
 {
  i=0;
  mboxfree=0;
  do
   {
       /* there is a free mailbox location */
       if((mbox[i].dest==FREE) && (mbox[i].src==FREE) && (mbox[i].len==ZERO))
                       {
               mboxfree=1;
               mbox[i].dest = dest_task;
               mbox[i].src = msg[0].src;
               mbox[i].len = msg[0].len;
               for (j=0;j<msg_len;j++)
```

```
        mbox[i].dat.string.s[j] = msg[0].dat.string.s[j];
                }
        i++;
        } while ((i<MBXSIZE) && !mboxfree);
 }
 EA=1;
}
/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : OS_CLEAR_MSG
*
* Function type : Task suspension system call
*
* Description   : This system call clears a message from mailbox for a task
*                  number
*
* Arguments     : task number
*
* Returns       : none
*
***************************************************************************
*/

void OS_CLEAR_MSG(uchar task_num)
{
 uchar i;

 EA=0;
  for(i=0;i<MBXSIZE;i++) {
   if((mbox[i].dest==task_num)&&(mbox[i].src!=FREE)&&(mbox[i].len!=ZERO))
        /* find relevant mailbox */

        mbox[i].dest = FREE;
        mbox[i].src = FREE;
        mbox[i].len = ZERO;
     }
   }
 EA=1;
}
/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : OS_CHECK_MSG
*
* Function type : Check message presence system call
*
```

```
* Description   : System call checks if there is a message for task in
*                 mailbox
*
* Arguments     : message, with the following structure
*                 struct letter{uchar dest,src,len;union dataformat dat;}
*
* Returns       : bit 1 if messge present
*                 bit 0 if message not present
*
**************************************************************************
*/
bit OS_CHECK_MSG(uchar task_num)
{
 if((mbox[task_num].dest==Running) && mbox[task_num].len>ZERO)
        return (1);
 else
        return (0);
}
/*
**************************************************************************
*/

/*
***************************************************************************
* Function name : OS_GET_MSG
*
* Function type : Get message from mailbox system call
*
```

```
* Description   : System call checks if there is a message for task in
* mailbox
*
* Arguments     : message, with the following structure
*                  struct letter{uchar dest,src,len;union dataformat dat;}
*
* Returns       : bit 1 if messge present
*                  bit 0 if message not present
*
**************************************************************************
*/
void OS_GET_MSG(struct letter xdata *msg)
{
uchar i,temp,j;
bit present;

 EA = 0;
 i=present=0;
 do
 {
       if((mbox[i].dest==Running) && mbox[i].len>ZERO && (task[Running].status1 & WAIT4M_F))
       /* If message was waiting for message, and then the message was placed*/

*/      later by a send message command */
 {
        present = 1;

       /* get message, clear mailbox and return to same task */

       msg[0].dest = mbox[i].dest;   /* task number of destination */
       msg[0].src = mbox[i].src;             /* task number of source */
       msg[0].len = temp = mbox[i].len;
               for (j=0;j<temp;j++)
                       msg[0].dat.string.s[j] = mbox[i].dat.string.s[j];

       mbox[i].dest = FREE; /* clear space in mailbox */
       mbox[i].src = FREE;
       mbox[i].len = ZERO;

       task[Running].status1 = ZERO;
       task[Running].status2 = ZERO;
 }
 i++;
 } while ( (i<MBXSIZE) && !present );
 EA = 1;
 }
/*
**************************************************************************
*/
/*
**************************************************************************
*
* Function name : OS_WAIT_MESSAGE
*
* Function type : task waiting for message system call
*
```

```
* Description    : this system call waits for a message
*
* Arguments      : message, with the following structure
*                  struct letter{uchar dest,src,len;union dataformat dat;}
* Returns        : none
*
***************************************************************************
*/
void OS_WAIT_MESSAGE(struct letter xdata *msg, uint ticks)
{
 uchar i,j,temp;
 uchar idata * idata internal;
 bit present,mboxfree;

 EA=0;

       RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case*/
/* there is a need for a voluntary task swap */
 SaveSFRs(&task[Running].rega);
 SaveBank0(&task[Running].reg0);
 RS0 = 0; /* use bank 0 */

       task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
       internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                   /* Current task's USED stack area is saved */
              task[Running].stack[i++] = *(internal++);
              } while (internal<=temp);
 i=present=0;
 do
 {
       if((mbox[i].dest==Running) && mbox[i].len>ZERO)
       /* If message already present */
 {
        present = 1;

/* get message, clear mailbox and return to same task */

       msg[0].dest = mbox[i].dest;   /* task number of destination */
       msg[0].src = mbox[i].src;             /* task number of source */
       msg[0].len = temp = mbox[i].len;
          for (j=0;j<temp;j++)
              msg[0].dat.string.s[j] = mbox[i].dat.string.s[j];

       mbox[i].dest = FREE; /* clear space in mailbox */
       mbox[i].src = FREE;
       mbox[i].len = ZERO;

       task[Running].status1 = ZERO;
       task[Running].status2 = ZERO;
   }
  i++;
 } while ( (i<MBXSIZE) && !present );
```

```
/* it executes this code if no message was present, */
/* hence it has to wait */
if (!present)
 {
 i=mboxfree=0;
 do
  {
       if((mbox[i].dest==FREE) && (mbox[i].src==FREE) && (mbox[i].len==ZERO))
       /* If mailbox available */
       {
       mboxfree=1;
       mbox[i].dest = Running; /* book mailbox by setting destination */
       mbox[i].src = FREE;
       mbox[i].len = ZERO;

       /* task change made here */
/* and the task then enters the waiting state */

 task[Running].status1 |= WAIT4M_F; /* mark task as waiting for message */
                                      /* flag used also by OS_GET_MSG */
 task[Running].timeout = ticks;
 task[Running].status2 &= ~PREEMP_F; /* mark task as NOT pre-emptied */
 V_TaskChange();                      /* a task switch.     */
 }
 i++;
 } while((i<MBXSIZE) && !mboxfree);
}
/* if no spare mailboxes, signal flag and return */
```

```
 if ( !present && !mboxfree )
        task[Running].status2 |= NO_MBOX_FREE_F; /* mark space for mailbox */

 EA = 1;

}

/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : OS_WAITT
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to go in the waiting state
*                 for a timeout period given by a defined number of
*                      RTOS ticks.
*
* Arguments     : ticks              Represents the number of ticks for which the
*                                    task will wait. Valid range for this parameter
*                                    is 1 to 65535.
*                                    A zero waiting time parameter is set to 1 by the
*                                    RTOS itself, since a zero effectively kills the
*                                    task, making it wait forever.
*
* Returns : None
*
**************************************************************************
*/

void OS_WAITT (uint ticks)
{
        uchar data i, temp;
                uchar idata * idata internal;
        EA = 0;

    RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
 RS0 = 0; /* use bank 0 */

        task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
        internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                   /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=temp);
        if (ticks == 0)
                ticks = 1;              /* Task's timeout variable is updated */
        task[Running].timeout = ticks;
```

```
 /* and the task then enters the      waiting state */
 task[Running].status2 &= ~PREEMP_F; /* mark task as NOT pre-emptied */

 V_TaskChange();                       /* Make a voluntary task change */
}
/*
****************************************************************************
*/

/*
****************************************************************************
*
* Function name : OS_WAITP
*
* Function type : Event-Waiting System call
*
* Description   : This system call is used by a task to wait for the end of
*                  its periodic interval.
*                  If the interval has already passed, task continues to
*                  execute.
*
* Arguments     : None
*
* Returns       : None
*
****************************************************************************
*/

void OS_WAITP(void)
{
       unsigned char i,temp;
               uchar idata * idata internal;

               EA = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a voluntary task swap */
 RS0 = 1; /* USE BANK 1 */
 SaveSFRs(&task[Running].rega);
 SaveBank0(&task[Running].reg0);
 RS0 = 0; /* RETURN TO BANK 0 */

       task[Running].stackptr = temp = SP;   /* Current task's SP is saved */
       internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */
       /* Current task's USED stack area is saved */
 i = 0;
 do {
               task[Running].stack[i++] = *(internal++);
               } while (internal<=temp);
 if (!(task[Running].status2 & WAIT4V_F))
 /* if periodic interval has not yet passed, as is generally the case */
 {
        task[Running].status1 |= WAIT4V_F;    /* set task to wait */
       task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */
        V_TaskChange();       /* make a voluntary context switch */
```

```
 }
 else                          /* if periodic time already elapsed, */
 {
 task[Running].status2 &= ~WAIT4V_F; /* clear waiting flag and */
 EA = 1; /* do nothing else and continue running */
        }
}

/*
***************************************************************************
*/
/*
***************************************************************************
*
* Function name : OS_PERIODIC
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to repeat its function
*                  every given
*                  number of RTOS ticks.
*
* Arguments     : ticks              Represents the length of the periodic
*                              interval in terms
*                              of RTOS ticks, after which the task repeats
*                              itself.
*                               Valid range for this parameter is 1 to 65535.
*
```

```
* Returns       : None
*
************************************************************************
*/

void OS_PERIODIC (uint ticks)
{
        EA = 0;
        if (!ticks) ticks = 1;        /* just a pre-caution */
        task[Running].interval_count = ticks;
/* Task's periodic interval count */
        task[Running].interval_reload = ticks;
/* and reload variables are initialised.      */
        EA = 1;
}

/*
************************************************************************
*/

/*
************************************************************************
*
* Function name : OS_WAITI
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to wait for a given event
*                 (interrupt). It identifies
*                 for which interrupt the task has to wait. Once identified
*                 - the task's appropriate
*                 flag is set and the task is set in the waiting state by causing
*                 a task swap - the task
*                 would wait indefinitely for the interrupt as its timeout variable
*                 would be set to 0
*                 (NOT_TIMING) either during initialisation of the RTOS or after
*                 expiry of its timeout period due to other
*                 prior invocations of wait-inducing system calls.
*
* Arguments     : intnum      Represents the interrupt number associated with
*                             the given
*                             interrupt for which the calling task intends to
*                             wait
*
* Returns       : None
*
************************************************************************
*/

void OS_WAITI(uchar int_num)
{
 unsigned char i,temp;
        uchar idata * idata internal;
        EA = 0;
```

```
        switch (int_num)
        {
        #if (!STAND_ALONE_ISR_00)
                case 0: /* Interrupt number 0 */
        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = EXT0_INT;      /* Task made to wait for */
                                                /* external interrupt 0 */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                  /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=temp);

        task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */
        V_TaskChange();                       /* Make a voluntary task change */
                break;
        #endif


        #if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
                case 1: /* Interrupt number 1 */
        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = TIM0_INT; /* Task made to wait for */
                                               /* timer 0 interrupt */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

        i = 0;
        do {            /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=temp);

        task[Running].status2 &= ~PREEMP_F;
 /* mark task as NOT pre-emptied */
```

```
         V_TaskChange();                       /* Make a voluntary task change */
                break;
        #endif
        #if (!STAND_ALONE_ISR_02)
                case 2:                        /* Interrupt number 2 */
        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = EXT1_INT;      /* Task made to wait for */
                                                        /* external interrupt 1 */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
 /* MAINSTACK is declared in STARTUP.A51 */
        i = 0;
        do {            /* Current task's USED stack area is saved */
                 task[Running].stack[i++] = *(internal++);
                 } while (internal<=temp);

        task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */

        V_TaskChange();                        /* Make a voluntary task change */
                break;
        #endif

        #if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
                case 3: /* Interrupt number 3 */
        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = TIM1_INT;      /* Task made to wait for */
                                        /* timer 1 interrupt */

 RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

        i = 0;
        do {            /* Current task's USED stack area is saved */
                 task[Running].stack[i++] = *(internal++);
                 } while (internal<=temp);

        task[Running].status2 &= ~PREEMP_F;
 /* mark task as NOT pre-emptied */
```

```
        V_TaskChange();                 /* Make a voluntary task change */
        break;
        #endif


        #if (!STAND_ALONE_ISR_04)
                case 4:                 /* Interrupt number 4 */\

        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = SER0_INT;      /* Task made to wait for */
                                                      /* serial port interrupt */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */
        i = 0;
        do {            /* Current task's USED stack area is saved */
                 task[Running].stack[i++] = *(internal++);
                 } while (internal<=temp);
        task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */
```

```
        V_TaskChange();                         /* Make a voluntary task change */
                break;
        #endif


        #if ( (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )
                case 5: /* Interrupt number 5 */
        task[Running].status1 |= WAIT4I_F; /* mark task as waiting int */
                task[Running].intnum = TIM2_INT;      /* Task made to wait for */
                                                /* timer 2 interrupt */
        RS0 = 1; /* use bank 1 */
/* store current task A,B,DPH,DPL SFRs and bank 0 registers just in case */
/* there is a need for a voluntary task swap */
        SaveSFRs(&task[Running].rega);
        SaveBank0(&task[Running].reg0);
        RS0 = 0; /* use bank 0 */

         task[Running].stackptr = temp = SP;
/* Current task's SP is saved */
         internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

        i = 0;
        do {    /* Current task's USED stack area is saved */
                 task[Running].stack[i++] = *(internal++);
                 } while (internal<=temp);
        task[Running].status2 &= ~PREEMP_F;
/* mark task as NOT pre-emptied */

        V_TaskChange();                        /* Make a voluntary task change */
                break;
         #endif
                default:
                EA = 1;
                break;
        }
}
/*
****************************************************************************
*/

/*
****************************************************************************
*
* Function name : OS_KILL_TASK
*
* Function type : Task Suspension System call
*
* Description   : This system call kills the current task, by putting it
*                 permanently waiting, such that
*                 it never executes again. It also clears any set
*                 waiting signals
*                 which the task might have.
*
* Arguments     : Task number to be killed
*
```

```
* Returns       : None
*
***************************************************************************/

void OS_KILL_TASK(uchar task_num)
{
unsigned char temp,i;
uchar data * idata qptr;
bit found;

       EA = 0;

 if (task[task_num].status2 & TASK_KILLED_F)
        {;}
 else
       {
 task[task_num].status2 |= TASK_KILLED_F;
       /* check if task waiting for some message */
       for(i=0;i<MBXSIZE;i++)
        {
         if (mbox[i].dest == task_num)
          {
               mbox[i].dest = FREE;
               mbox[i].src = FREE;
               mbox[i].len = ZERO;
               }
        }
       if (task_num == Running)
/* if task happens to be the present running one */
        {
        task[Running].status1 = ZERO;
 /* Killed by clearing its flags */
        task[Running].status2 = ZERO;
 /* Killed by clearing its flags */
        task[Running].timeout = NOT_TIMING;
 /* setting it to wait forever */
        task[Running].interval_count = ZERO;
 /* Periodic interval count 0*/
        task[Running].interval_reload = ZERO;
/* Periodic interval count 0 */
        task[Running].priority = LOWEST; /* lowest priority setting */
       task[Running].status2 |= TASK_KILLED_F;
               V_TaskChange(); /* and then cause a task switch. */
        }
/* Otherwise, search the ready queue, and if it is there, simply change its
       number to that of the idle task */
     found = 0;
       qptr = ReadyQ;
       while (qptr <= ReadyQTop)
        {
               if (*qptr == task_num)
               { *qptr = IDLE_TASK; found = 1;}
               qptr++;
              }
```

```
/* The Ready Queue is ALWAYS sorted before ATTEMPTING any task change */
if ((ReadyQTop != ReadyQ) && found)
        {                              /* the queue is sorted such */
                qptr = ReadyQ;         /* that the idle task   */
                while (qptr < (ReadyQTop-1)) /* ends at ReadyQTop */
                {
                        if (*qptr == IDLE_TASK)
                        {
                                temp = *qptr;
                                *qptr = *(qptr + 1);
                                *(qptr + 1) = temp;
                        }
                qptr++;
                }
 ReadyQTop--;           /* to eliminate double IDLE_TASKS */
         }
 if(!found) /* if task is waiting for some event or timeout */
         {
         task[task_num].status1 = ZERO;
/* Killed by clearing its flags */
         task[task_num].status2 = ZERO;
/* Killed by clearing its flags */
         task[task_num].catalog = IDLE_TASK;
/* changing id to IDLE_TASK */
         task[task_num].timeout = NOT_TIMING;
/* setting it to wait forever */
         task[task_num].interval_count = ZERO;
```

```
/* Periodic interval count 0 */
        task[task_num].interval_reload = ZERO;
/* Periodic interval count 0 */
        task[task_num].priority = LOWEST; /* Periodic interval count 0 */
 task[task_num].status2 |= TASK_KILLED_F;
        }

 }
 EA = 1;
}

/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : V_TaskChange
*
* Function type : Context Switch (Internal function)
*
* Description   : This function is used to perform a voluntary context
*                 switch i.e. task swapping
*
* Arguments     : none
*
* Notes         :
*
* Returns       : None
*
**************************************************************************
*/

void V_TaskChange (void)
{
       uchar data i, temp;
       uchar idata * idata internal;
       uchar data * idata qtask;
       uchar data * idata qptr;

       TinQFlag = 0;

       internal =      MAINSTACK;
/* MAINSTACK is the address of the start of */
                              /* the main internal stack defined in */
                              /* STARTUP.A51 */
/* The Ready Queue is ALWAYS sorted before ATTEMPTING any task change */
if (ReadyQTop != ReadyQ)
 { /* the queue is sorted such that */
       qptr = ReadyQTop; /* the highest priority task       */
              while (qptr > ReadyQ) /* becomes the first in ready queue */
              {
                     if (task[*qptr].priority > task[*(qptr - 1)].priority)
                     {
```

```
                              temp = *qptr;
                              *qptr = *(qptr - 1);
                              *(qptr - 1) = temp;
                      }
       qptr--;
               }
        }
        Running = ReadyQ[0];  /* set the new task as running */
/* READY queue is shifted down by one position only after a task change */

       qtask = ReadyQ;
 qptr = qtask + 1;
       while (qtask <= ReadyQTop)
       {
               *qtask++ = *qptr++;
       }
       ReadyQTop--;   /* Pointer to last task in queue is decremented */

       if (ReadyQTop < ReadyQ)       /* Ensure that this pointer is never */
               ReadyQTop = ReadyQ;/* below the start of the READY queue */

 /* The new running task's USED stack area is copied to internal RAM */
 temp = task[Running].stackptr;
       internal = MAINSTACK;
 i=0;
 do    {
               *(internal++) = task[Running].stack[i++];
       } while (internal<=temp);
       SP = temp;     /* The new running task's SP is restored */

 if (task[Running].status1 & WAIT4I_F)
 /* if new task was waiting for interrupt, */
       {
       task[Running].status1 &= ~WAIT4I_F;
/* then clear interrupt flag */
       /* Get the new tasks registers which were stored externally */
       RS0 = 1; /* USE BANK 1 */
       RecallBank0(&task[Running].reg0);
       RS0 = 0; /* RETURN USING BANK 0 */
       POP5(); /* starts other task here */
       }
 else if (task[Running].status2 & PREEMP_F)
 /* if new task was pre-emptied before, */
       {
       task[Running].status2 &= ~PREEMP_F;
/* then clear pre-emptied flag */
       /* Get the new tasks registers which were stored externally */
       RS0 = 1; /* USE BANK 1 */
       RecallBank0(&task[Running].reg0);
       RS0 = 0; /* RETURN USING BANK 0 */
       POP5(); /* starts other task here */
       }

 else if (task[Running].status2 & FIRST_TIME_F)
       {                      /* if new task running for the 1st time, */
       task[Running].status2 &= ~FIRST_TIME_F;
```

```
/* then clear the flag */
         SP = temp; /* The new running task's SP is restored */
        }

 else
        {
        RS0 = 1; /* USE BANK 1 */
        RecallBank0(&task[Running].reg0);
        RecallSFRs(&task[Running].rega);
        RS0 = 0; /* RETURN USING BANK 0 */
        }
        EA = 1;
}

/*
***************************************************************************
*/

/*
***************************************************************************
*
* Function name : PE_TaskChange
*
* Function type : Context Switch (Internal function)
*
* Description   : This function is used to perform a forced or pre-emtive
*                  i.e. context switch or task swapping
*
```

```
* Arguments     : none
*
*
* Notes         : This procedure is called from the timer tick interrupt,
*                 there would be 5 registers pushed on the stack, saved
*                 while the current task was running.
*                 Push A, B, DPH, DPL and PSW
*
*                 Comes here ONLY from an Interrupt
*                             (Tick Timer or other)
* Returns       : None
*
**************************************************************************
*/

void PE_TaskChange (void) using 1
{
        uchar data i, temp;
        uchar idata * idata internal;
        uchar data * idata qtask;
        uchar data * idata qptr;

        TinQFlag = 0;

/* The Ready Queue is ALWAYS sorted before ATTEMPTING any task change */

if (ReadyQTop != ReadyQ)
 { /* the queue is sorted such that */
        qptr = ReadyQTop; /* the highest priority task       */
                while (qptr > ReadyQ) /* becomes the first in ready queue */
                {
                        if (task[*qptr].priority > task[*(qptr - 1)].priority)
                        {
                                temp = *qptr;
                                *qptr = *(qptr - 1);
                                *(qptr - 1) = temp;
                        }
                qptr--;
                }
         }
/* The first task in the READY queue has a higher priority than the current */
/* one, therefore the current task is PRE-EMPTIED, Queue shifted down */
/* the current task is placed at the top of the Ready Queue again */
/* so that */
/* it can continue to run when its priority allows it to and a */
/* new task is set to run */

        task[Running].status2 |= PREEMP_F; /* mark old task as pre-emptied */
        temp = Running;

/* NOW WORK WITH THE NEW TASK */
         internal = MAINSTACK; /* MAINSTACK is the address of the start */
                              /* of main stack defined in STARTUP.A51 */
         Running = ReadyQ[0];  /* set the new task as running */
```

```
/* READY queue is shifted down by one position only after a task change */

        qtask = ReadyQ;
 qptr = qtask + 1;
        while (qtask <= ReadyQTop)
        {
                *qtask++ = *qptr++;
        }
        ReadyQTop--;   /* Pointer to last task in queue is decremented */
        if (ReadyQTop < ReadyQ)       /* Ensure that this pointer is never */
                ReadyQTop = ReadyQ; /* below the start of the READY queue */

 ReadyQTop++;
         *ReadyQTop = temp; /* the old task is placed in the ready queue. */

 /* The new running task's USED stack area is copied to internal RAM */
 /* and the stack pointer adjusted accordingly */
    temp = task[Running].stackptr;
        internal = MAINSTACK;
 i=0;
 do     {
                *(internal++) = task[Running].stack[i++];
        } while (internal<=temp);
        SP = temp;     /* The new running task's SP is restored */

 if (task[Running].status1 & WAIT4I_F)
 /* if new task was waiting for interrupt, */
        {
        task[Running].status1 &= ~WAIT4I_F;
/* then clear interrupt flag */
        /* Get the new tasks registers which were stored externally */
        RecallBank0(&task[Running].reg0);
        POP5I(); /* starts other task here */
        }

        else if (task[Running].status2 & PREEMP_F)
        /* if new task was pre-emptied before, */
        {
        task[Running].status2 &= ~PREEMP_F;
 /* then clear pre-emptied flag */
        /* Get the new tasks registers which were stored externally */
        RecallBank0(&task[Running].reg0);
        POP5I(); /* starts other task here */
        }

 else
        {
        if (task[Running].status2 & FIRST_TIME_F)
                { /* if new task running for the 1st time, */
                task[Running].status2 &= ~FIRST_TIME_F;
 /* then clear the flag */
        POP0I(); /* starts other task here */
 }

 else
        { /* if new task had voluntarily given up before, do nothing */
        /* Get the new tasks registers */
```

```
        RecallBank0(&task[Running].reg0);
        RecallSFRs(&task[Running].rega);
        POP0I(); /* starts the other task here */
        }
 }

 EA = 1;
}
/*
****************************************************************************
*/

/*
****************************************************************************
*
* Function name : Xtra_Int_0
*
* Function type : Interrupt Service Routine
*
* Description   : This is the external 0 interrupt ISR whose associated
*                 interrupt number is 0.
*
* Arguments     : None
*
* Returns       : None
*
***************************************************************************
*/
```

```
#if (!STAND_ALONE_ISR_00)
void Xtra_Int_0 (void) interrupt 0 using 1
{
        uchar idata * idata internal;
        uchar data i,k;

       EA = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler after the interrupt */
/* and are saved as part of the task stack */
 SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

 task[Running].stackptr = k = SP;
 /* Current task SP is saved pointing to PSW which is the last one */
 /* push on stack after the interrupt */
       internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                  /* Current task's USED stack area is saved */
              task[Running].stack[i++] = *(internal++);
              } while (internal<=k);

       Xtra_Int(EXT0_INT);
/* Passes EXT0_INT for identification purposes */
}
#endif

/*
**************************************************************************
*
* Function name : RTOS_Timer_Int
*
* Function type : Scheduler Interrupt Service Routine
*
* Description   : This is the RTOS scheduler ISR. It generates system ticks
*                  and calculates any remaining
*                      waiting and periodic interval time for each task.
*
* Arguments     : None
*
* Returns       : None
*
**************************************************************************
*/

#if (TICK_TIMER == 0)
       /* If Timer 0 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 1 using 1
{
        uchar idata * idata internal;
        uchar data k;          /* For the 8051, Timer 0 is often */
        uchar data * idata q; /* used for scheduling.                      */
        bit bdata On_Q;
/* After an interrupt, the SP is incremented by 5 by the */
```

```
/* compiler to PUSH ACC,B,DPH,DPL and PSW */
/* These are popped back before returning from the interrupt */

        TH0 = HiByte(BASIC_TICK);    /* Timer registers reloaded */
        TL0 = LoByte(BASIC_TICK);

#elif (TICK_TIMER == 1)       /* If Timer 1 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 3 using 1
{
        uchar idata * idata internal;
        uchar data k;          /* For the 8051, Timer 1 can be used */
        uchar data * idata q; /* for scheduling, provided it is not */
        bit bdata On_Q; /* needed as UART baud rate generator */

/* After an interrupt, the SP is incremented by 5 by the */
/* compiler to PUSH ACC,B,DPH,DPL and PSW */
/* These are popped back before returning from the interrupt */
/* PSW is also pushed bexause of the using 1 command */
        TH1 = HiByte(BASIC_TICK);    /* Timer registers reloaded */
        TL1 = LoByte(BASIC_TICK);

#elif (TICK_TIMER == 2)       /* If Timer 2 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 5 using 1
{
        uchar idata * idata internal;
        uchar data i,k;              /* For the 8032, Timer 2 is used */
        uchar data * idata q;        /* for scheduling.                   */
        bit bdata On_Q;

/* After an interrupt, the address of the next instruction of the */
/* current task is push on stack (low then high byte). Then SP */
/* is further incremented by 5 by the */
/* compiler to PUSH ACC,B,DPH,DPL and PSW */
/* Internal stack map at this stage */
/* High stack RAM */
/* PSW <-- SP points to here */
/* DPL */
/* DPH */
/* B */
/* ACC */
/* High byte return address */
/* Low byte return address */
/* Low stack RAM */

/* These are normally popped back BEFORE returning from the */
/* interrupt IF the TaskChange function is not called. */

        TF2 = 0;              /* Timer 2 interrupt flag is cleared */

#endif


/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
```

```
/* after the interrupt */
/* and are saved as part of the task stack */
        SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

        task[Running].stackptr = k = SP;
        /* Current task SP is saved pointing to PSW which is the last one */
        /* push on stack after the interrupt */
                internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                   /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
        } while (internal<=k);
         for (k = 0; k < NOOFTASKS; k++)
         {

/* check those tasks that are PERIODIC */
                if (task[k].interval_count != ZERO) /* Updates the tasks */
                {                              /* periodic intervals. */
                        task[k].interval_count--;
                        if (task[k].interval_count == ZERO)
                        {
                task[k].interval_count = task[k].interval_reload;
                                        if (task[k].status1 & WAIT4V_F)
                                {

/* If periodic interval       */
/* has elapsed and the        */      task[k].status1 &= ~WAIT4V_F;
/* task has been waiting      */      q = ReadyQ;
/* for this to occur, the     */      On_Q = 0;
/* task is placed in the      */      while (q <= ReadyQTop)
/* READY queue, if it is      */ {
/* verified that the task     */              if (k == *q)
/* does not already reside    */              {
/* in the queue, as now       */              On_Q = 1;
/* the task no longer         */ break;
/* requires to wait.          */               }
/* This can happen due to BAD programming, */
/* with a task taking much longer */
/* to execute than the periodic interval requested */
                                      q++;
                                              }

                                if      (!On_Q)
                                        {
                                        ReadyQTop++;
                                        *ReadyQTop = k;
/* put in ready queue */
                                if (task[k].priority > task[Running].priority)
                                        TinQFlag = 1;
                                        /* mark new higher priority task in Q */
                                        /* setting flag to start pre-emption */
                                        }
                        }
```

```
/* If however the task was not waiting for this event */
/* the task is not placed in the ready queue, but a flag is */
/* set to indicate that the periodic time has already passed */

                      else
                              task[k].status1 |= WAIT4V_F;
                       }
              }

/* Now, check for any tasks waiting for a timeout */
              if (task[k].timeout != NOT_TIMING)
        {                                      /* Updates the tasks' */
                      task[k].timeout--; /* timeout variables. */
                      if (task[k].timeout == ZERO)
              {
                              ReadyQTop++; /* If a waiting task's */
                              *ReadyQTop = k; /* timeout elapses */
                      /* Clear flags just in case it had been */
                              task[k].status1 &= ~WAIT4M_F;
/* waiting message + timeout */
/* need to take care of message case, (timeouts before message arrives) */
                              task[k].status1 &= ~WAIT4S_F;
/* waiting semaphore + timeout*/

                      /* the task is placed in the ready queue. */
        if (task[k].priority > task[Running].priority) TinQFlag = 1;
                      /* set flag to start pre-emption */
                              }
              }
```

```
                /* If any new task higher priority task was placed in */
               /* in the ready queue, then we need to do a */
        /* pre-emptive task switch by calling TaskChange, Option 0 */
        }

        if (TinQFlag)
        {
               PE_TaskChange();
/* Force a pre-emptive task change if required */
/* Note that the pushed task registers would still be on the saved */
/* stack at */
/* this point and would be popped back when task is put into action again */
        }

/* else return to original running task, popping the pushed registers */
/* automatically by the KEIL compiler. */
EA = 1;
}

/*
*************************************************************************
*/

/*
*************************************************************************
*/

/*
*************************************************************************
*
* Function name : Xtra_Int_1
*
* Function type : Interrupt Service Routine
*
* Description   : This is the Timer 0 ISR whose associated interrupt number
*                 is 1.
*                 It is only enabled in
*                 case an 8032 microcontroller is being used in combination
*                 with an EEPROM. The reason
*                 being is that without an EEPROM Timer 0 is not available
*                 on the 8032 and in case of
*               the 8051 Timer 0 is already being used as the RTOS
*                 scheduler.
*                 It is also available if using the version 2 monitor ROM.
*
* Arguments     : None
*
* Returns       : None
*
*************************************************************************
*/

#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )

/* Timer 0 interrupt is usually used for RTOS on the basic 8051 */
/* For the FLT-32 8032, it can only be used with the modified monitor */
```

```
/* or user eprom */
/* it is used for the single step in the old version monitor eprom */

void Xtra_Int_1 (void) interrupt 1 using 1
{
         uchar idata * idata internal;
         uchar data i,k;

        EA = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
/* after the interrupt */
/* and are saved as part of the task stack */
        SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

        task[Running].stackptr = k = SP;
        /* Current task SP is saved pointing to PSW which is the last one */
        /* push on stack after the interrupt */
               internal = MAINSTACK;
        /* MAINSTACK is declared in STARTUP.A51 */

        i = 0;
        do {                  /* Current task's USED stack area is saved */
               task[Running].stack[i++] = *(internal++);
               } while (internal<=k);

        Xtra_Int(TIM0_INT);
/* Passes TIM0_INT for identification purposes */
}

#endif

/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : Xtra_Int_2
*
* Function type : Interrupt Service Routine
*
* Description   : This is the external 1 interrupt ISR whose associated
*                 interrupt number is 2.
*
* Arguments     : None
*
* Returns       : None
*
**************************************************************************
*/
#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2 (void) interrupt 2 using 1
{
```

```
         uchar idata * idata internal;
         uchar data i,k;

        EA = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
/* after the interrupt */
/* and are saved as part of the task stack */
        SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */
        task[Running].stackptr = k = SP;
        /* Current task SP is saved pointing to PSW which is the last one */
        /* push on stack after the interrupt */
                internal = MAINSTACK;
/* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                   /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=k);
        Xtra_Int(EXT1_INT);
/* Passes EXT1_INT for identification purposes */
}
#endif
/*
****************************************************************************
*/
```

```
/*
**************************************************************************
*
* Function name : Xtra_Int_3
*
* Function type : Interrupt Service Routine
*
* Description   : This is the Timer 1 ISR whose associated interrupt number
*                  is 3.
*
* Arguments     : None
*
* Returns       : None
*
**************************************************************************
*/
#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )

void Xtra_Int_3 (void) interrupt 3 using 1
{
        uchar idata * idata internal;
        uchar data i,k;
       EA = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
/* after the interrupt */
/* and are saved as part of the task stack */
       SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

       task[Running].stackptr = k = SP;
       /* Current task SP is saved pointing to PSW which is the last one */
        /* push on stack after the interrupt */
              internal = MAINSTACK;         /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {                  /* Current task's USED stack area is saved */
              task[Running].stack[i++] = *(internal++);
              } while (internal<=k);

       Xtra_Int(TIM1_INT);
/* Passes TIM1_INT for identification purposes */
}

#endif
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : Xtra_Int_4
*
* Function type : Interrupt Service Routine
*
```

```
* Description   : This is the serial port ISR whose associated interrupt
*                 number is 4.
*
* Arguments     : None
*
* Returns       : None
*
**************************************************************************
*/
#if (!STAND_ALONE_ISR_04)
void Xtra_Int_4 (void) interrupt 4 using 1
{
       uchar idata * idata internal;
       uchar data i,k;

       EA = 0;
/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
/* after the interrupt */
/* and are saved as part of the task stack */
       SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */
       task[Running].stackptr = k = SP;
       /* Current task SP is saved pointing to PSW which is the last one */
       /* push on stack after the interrupt */
              internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

       i = 0;
       do {                   /* Current task's USED stack area is saved */
              task[Running].stack[i++] = *(internal++);
              } while (internal<=k);

       Xtra_Int(SER0_INT);
/* Passes SER0_INT for identification purposes */
}
#endif
/*
**************************************************************************
*/

/*
**************************************************************************
*
* Function name : Xtra_Int_5
*
* Function type : Interrupt Service Routine
*
* Description   : This is the Timer 2 ISR whose associated interrupt
*                 number is 5.
*
* Arguments     : None
*
* Returns       : None
*
```

```
************************************************************************
*/
#if ( (CPU == 8032) && (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )

void Xtra_Int_5 (void) interrupt 5 using 1
{
         uchar idata * idata internal;
         uchar data i,k;

        EA = 0;
 TF2 = 0;

/* store current task bank 0 registers just in case there is */
/* a need for a pre-emptive task swap */
/* A,B,DPH,DPL and PSW are pushed on stack by the compiler */
/* after the interrupt */
/* and are saved as part of the task stack */
        SaveBank0(&task[Running].reg0); /* store R0 - R7 bank 0 */

        task[Running].stackptr = k = SP;
        /* Current task SP is saved pointing to PSW which is the last one */
        /* push on stack after the interrupt */
                internal = MAINSTACK; /* MAINSTACK is declared in STARTUP.A51 */

 i = 0;
 do {           /* Current task's USED stack area is saved */
                task[Running].stack[i++] = *(internal++);
                } while (internal<=k);
        Xtra_Int(TIM2_INT);
/* Passes TIM2_INT for identification purposes */
}

#endif
/*
************************************************************************
*/

/*
************************************************************************
*
* Function name : Xtra_Int
*
* Function type : Interrupt Handling (Internal function)
*
* Description   : This function performs the operations required by the
*                 previous ISRs.
*
* Arguments     : task_intflag        Represents the flag mask for a given
*                                     interrupt against which the
*                                             byte storing the flags of each task
*                                     will be compared in order to
*                                             determine whether any task has been
*                                     waiting for the interrupt in
*                                             question.
*
```

```
* Returns        : None
*
***************************************************************************
*/
void Xtra_Int (uchar current_intnum) using 1
{
        uchar data k;

        for (k = 0; k < NOOFTASKS; k ++)
        {
                if (task[k].intnum == current_intnum)
                {
                 task[k].intnum = NO_INTERRUPT;
                 task[k].status1 &= ~WAIT4I_F;
/* mark task as not waiting int */
//               task[k].timeout = ONE; /* If it found that a task */
                 if (task[k].priority > task[Running].priority) TinQFlag = 1;

                 task[k].timeout = NOT_TIMING;
/* If it found that a task */
                 ReadyQTop++;            /* has been waiting for the */
                 *ReadyQTop = k;         /* given interrupt, it no     */
                }                        /* longer requires to wait */
        }                               /* and is therefore placed */
                                /* on the READY queue.  */
                                 /* It will be handled at the next tick */
 EA = 1;
}
```

```
/*
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
*/
```

MagnOS.h

```
*************************************************************************
*                              RTOS KERNEL HEADER FILE
*
* For use with MagnOS_V01.C - Co-Operative RTOS written in C
*              by Ing. Paul P. Debono
*                              Use with the 8051 family of microcontrollers
*
* File         : MagnOS_V01.H
* Revision     : 1
* Date         : February 2006
* By           : Paul P. Debono
*
*                       B. Eng. (Hons.) Elec. Course
*                       University Of Malta
*
*************************************************************************
*/
#include "Parameters.H"

/*
*************************************************************************
*                              DATA TYPE DEFINITIONS
*************************************************************************
*/

typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned long ulong;

/*
*************************************************************************
*                      STRUCTURE AND UNION DEFINITIONS
*************************************************************************
*/

#define DATASIZE        16

union dataformat{struct{ulong HI1,LO1,HI0,LO0;}dblwords;
                 struct{uint Hi3,Lo3,Hi2,Lo2,Hi1,Lo1,Hi0,Lo0;}words;
                 struct{uchar hi7,lo7,hi6,lo6,hi5,lo5,hi4,lo4,
                              hi3,lo3,hi2,lo2,hi1,lo1,hi0,lo0;}bytes;
                 struct{char s[DATASIZE];}string;};

struct letter{uchar dest,src,len;union dataformat dat;};

struct task_param {    /* 13 bytes + 13 registers + stack per task */
       uchar catalog;         /* task id */
       uchar status1;         /* status flags, see below for details */
       uchar status2;         /* status flags, see below for details */
       uchar priority;        /* priority number, low = high priority */
       uchar semaphore;               /* counting semaphore for each task */
       uchar resource;                /* resource number required */
       uchar stackptr;        /* stack pointer SP storage location */
       uchar intnum;          /* task waiting for this interrupt number */
       uint timeout;          /* task waiting for this timeout in ticks, */
                                      /* 0 = not waiting */
```

```
         uint interval_count; /* time left to wait for this periodic */
                                       /* interval task in ticks */
         uint interval_reload;/* periodic tick interval reload value */
        uchar rega;           /* registers storage area, ready for context */
                              /* switching use */
        uchar regb;
        uchar rdph;
        uchar rdpl;
        uchar rpsw;
        uchar reg0;
        uchar reg1;
        uchar reg2;
        uchar reg3;
        uchar reg4;
        uchar reg5;
        uchar reg6;
        uchar reg7;
        char stack[STACKSIZE]; /* stack storage area */
                };

/*
**************************************************************************
*/

/*
**************************************************************************
*/
/* The MAINSTACK pointer variable points to the start pointer in */
/* hardware stack and should be defined in STARTUP.A51 */
extern idata unsigned char MAINSTACK[STACKSIZE];

/* Functions written in assembly language, found in MAGNOS_A01.A51 */
extern void SaveBank0(uchar xdata * Pointer);
extern void RecallBank0(uchar xdata * Pointer);
extern void SaveSFRs(uchar xdata * Pointer);
extern void RecallSFRs(uchar xdata * Pointer);
extern void POP5(void), POP0(void);
extern void POP5I(void), POP0I(void);

/*
**************************************************************************
*                              FUNCTION PROTOTYPES
**************************************************************************
*
*
* The following RTOS system calls do not receive any parameters :
* ----------------------------------------------------------------
*/

void OS_RTOS_GO (void);        // Starts the RTOS running with prioities if
                                       // required
void OS_WAITP (void);         // Waits for end of task's periodic interval
uchar OS_RUNNING_TASK_ID(void); // Returns the number of the currently
                                // executing (running) task

/* The following commands are simply defined as MACROS below
        OS_CPU_IDLE()  Set the microprocessor into a sleep mode
                              (awake every interrupt)
        OS_CPU_DOWN()  Switch off microprocessor, activate only by
```

```
 hardware reset
 OS_PAUSE_RTOS()        Disable RTOS, used in a stand alone ISR
 OS_RESUME_RTOS()       Re-enable RTOS, used in a stand alone ISR
*/

/*
* The following RTOS system calls do receive parameters :
* -------------------------------------------------------
*/

void OS_INIT_RTOS (uchar iemask);     // Initialises all RTOS variables
void OS_WAITI (uchar intnum);         // Waits for an event (interrupt)
void OS_WAITT (uint ticks);           // Waits for a timeout period given
                                      // by a defined number of ticks
uchar OS_CHECK_TASK_PRIORITY(uchar task_num);
                                      // Gets the requested task priority
void OS_CHANGE_TASK_PRIORITY(uchar task_num, uchar new_prio);
                                      // Sets the requested task priority
void OS_RELEASE_RES (uchar Res_Num);
void OS_WAIT4RES (uchar Res_Num,uint ticks);

void OS_SEND_MSG(struct letter xdata *msg);
void OS_CLEAR_MSG(uchar task_num);
bit OS_CHECK_MSG(uchar task_num);
void OS_GET_MSG(struct letter xdata *msg);
void OS_WAIT_MESSAGE(struct letter xdata *msg, uint ticks);

uchar OS_CHECK_TASK_SEMA4(uchar task_num);
                              // Gets the requestes task semaphore
void OS_SEMA4MINUS (uchar task_num, uchar units);
                              // Subtracts units from a semaphore
                                      // Causes task change if semapphore=0
void OS_SEMA4PLUS (uchar task_num, uchar units);
                              // Adds units to a semaphore
void OS_WAIT4SEM (uint ticks); // Waits for a signal to arrive
                              // within a given number of ticks
void OS_PERIODIC (uint ticks); // Sets task to behave periodically
                                      // every given number of ticks
void OS_CREATE_TASK (uchar tasknum, uint taskadd, uchar priority);
                                      // Creates a task
void OS_KILL_TASK (uchar tasknum);// Kills the selected task

/* The following commands are simply defined as MACROS below
       OS_WAITT_A(M,S,ms)     Absolute WAITT for minutes, seconds, msecs
       OS_WAITS_A(M,S,ms)     Absolute WAITS for minutes, seconds, msecs
       OS_PERIODIC_A(M,S,ms)  Absolute PERIODIC for minutes, seconds, msecs

       OS_WAIT4MSG(m,t)       Waits for message and gets message,
                              clearing mailbox
*/
/*
***************************************************************************
*/

#define STAND_ALONE_ISR_00 0   // EXT0 - set to 1 if using this interrupt
                              // as a stand alone ISR
```

```
#define STAND_ALONE_ISR_01 0   // TIM0 - set to 1 if using this interrupt
                              // as a stand alone ISR
#define STAND_ALONE_ISR_02 0   // EXT1 - set to 1 if using this interrupt
                              // as a stand alone ISR
#define STAND_ALONE_ISR_03 0   // TIM1 - set to 1 if using this interrupt
                              // as a stand alone ISR
#define STAND_ALONE_ISR_04 0   // SER0 - set to 1 if using this interrupt
                              // as a stand alone ISR
#define STAND_ALONE_ISR_05 0   // TIM2 - set to 1 if using this interrupt
                              // as a stand alone ISR
/*
**************************************************************************
*/

/*
**************************************************************************
*                              RTOS TIMING DEFINITIONS
**************************************************************************
*/

#define MSEC10 9216UL                 // In theory 921.6 counts represent
 // 1 msec assuming an
                                             // 11.0592 MHz crystal.
#define TICKS_PER_SEC   (1000 / TICKTIME)
                                             // Ensure that TICKTIME's value is
                                                    // chosen such that this
#define TICKS_PER_MIN   (60000 / TICKTIME)     // quotient and hence all the
                                             // following quotients result
                                                    // in an integer. In theory,
                                             // the maximum
                                                    // value of TICKTIME
                                                           // is given by the value
                                             // corresponding to CLOCK = 65535
#define CLOCK ((TICKTIME * MSEC10)/10) // i.e. approx. 70-72 - However
                                             // respecting the condition
#define BASIC_TICK      (65535 - CLOCK + 1)    // above, max. acceptable
                                                    // TICKTIME = 50 msecs.
// Hence all suitable values are:
// 1, 2, 4, 5, 8, 10, 20, 25, 40, 50
// For reliable time-dependent
// results a value of 10 or
// above is recommended depending
// upon the application

/* OTHER #defines */
#define MBXSIZE         20
#define FREE 0xFF       /* mailbox location is available */
#define EMPTY 0xFF      /* mailbox location is available */
#define NOOFRESOURCES   8

#define ZERO            0
#define ONE             1
#define NOT_TIMING             0        // An indefinite period of waiting time in the RTOS
                              // is given by a value of 0
#define NO_INTERRUPT    0xFF
```

```
// signifies task is not waiting for any interrupt event
#define LOWEST          0x00 /* lowest priority number */
#define HIGHEST         0xFF /* highest priority number */
#define MAXSEM          0xFF /* maximum number of units in a semaphore */

#define HiByte(intval)  (##intval)/256;
#define LoByte(intval)  (##intval)%256;
/* or you may use */
// #define HiByte(intval)  (unsigned char)(((##intval)& 0xFF00)>>8)
// #define LoByte(intval)  (unsigned char)((##intval)  & 0x00FF)

/*
***************************************************************************
*/

/*
***************************************************************************
*                               RTOS MACROS
***************************************************************************
*/

#define OS_CPU_IDLE()           PCON  |= 0x01
// Sets the microprocessor in idle mode
#define OS_CPU_DOWN()           PCON  |= 0x02
        // Sets the microprocessor in power-down mode

#if (TICK_TIMER == 0)
       #define OS_PAUSE_RTOS() EA = ET0 = TR0 = 0
       #define OS_RESUME_RTOS() TR0 = ET0 = EA = 1
```

```
#elif (TICK_TIMER == 1)
       #define OS_PAUSE_RTOS() EA = ET1 = TR1 = 0
       #define OS_RESUME_RTOS() TR1 = ET1 = EA = 1

#elif (TICK_TIMER == 2)
 #define OS_PAUSE_RTOS() EA = ET2 = TR2 = 0
 #define OS_RESUME_RTOS() TR2 = ET2 = EA = 1

#endif

/*
***************************************************************************/

/*
**************************************************************************
*              COMPILE-TIME ERROR TRAPPING
**************************************************************************
*/

#if (CPU != 8032) && (CPU != 8051)
       #error Invalid CPU Setting
#endif

#if (NOOFTASKS > 254)
#error Number of tasks is too big. The ReadyQ can store up to 254 tasks
#endif

#if ((TICKTIME * 110592 / 120) > 65535)
       #error Tick time value exceeds valid range for timer counter setting
#endif

#if ((TICKTIME * 110592 / 120) < 65535) && ((1000 % TICKTIME) != 0)
#error Undesirable TICKTIME setting (1, 2, 4, 8, 10, 20, 25, 40, 50 ms)
#endif

#if (CLOCK > 65535)
#error Timer counter setting exceeded valid range. Check TICKTIME and MSEC
#endif

/*
**************************************************************************
*/

/*
**************************************************************************
*                             TASK-RELATED DEFINITIONS
**************************************************************************
*/

/* Interrupt numbers, used for tasks waiting for some interrupt event */
#define EXT0_INT 0x00          // External 0 Interrupt number 0
#define TIM0_INT 0x01          // Timer 0 Interrupt number 1
#define EXT1_INT 0x02          // External 1 Interrupt number 2
#define TIM1_INT 0x03          // Timer 1 Interrupt number 3
#define SER0_INT 0x04          // Serial Interrupt number 4
#define TIM2_INT 0x05          // Timer 2 Interrupt number 5
```

```
#define IDLE_TASK NOOFTASKS
// Main endless loop in application given a task
// number equal to NOOFTASKS

/*
***************************************************************************
*/

/*
***************************************************************************
*                      ENHANCED EVENT-WAITING ADD-ON MACROS
****************************************************************************
*
* These macros perform the same functions of the WAITT, WAITS and
* PERIODIC calls
* but rather than ticks they accept absolute time values as parameters in
* terms of days, hours, minutes, seconds and millisecs
* This difference is denoted by the _A suffix - eg. WAITT_A() is the
* absolute-time version of WAITT()
*
* Range of values accepted, (maximum 65535 TICKTIMES):
*
* Using a TICKTIME of 1 msec : 1 msecs - 1 min, 5 secs, 535 msecs
* Using a TICKTIME of 10 msec : 10 msecs - 10 mins, 55 secs, 350 msecs
* Using a TICKTIME of 50 msec : 50 msecs - 54 mins, 36 secs, 750 msecs

* If the conversion from absolute time to ticks results in 0
* (all parameters
* being 0 or overflow) this result is only accepted by WAITS()
* by virtue of how
* the WAITT(), WAITS() and PERIODIC() calls were written.
* In the case of the
* WAITT() and PERIODIC() calls the tick count would automatically be
* changed to 1
* meaning an interval of eg. 50 msecs in case the TICKTIME is defined to
* be 50 msecs
*
* Liberal use of parentheses is made in the following macros in case the
* arguments might be expressions
*
****************************************************************************/

#define TPM(M) (TICKS_PER_MIN*(##M))
#define TPS(S) (TICKS_PER_SEC*(##S))
#define TPMS(ms) ((##ms)/TICKTIME)

#define OS_WAITT_A(M,S,ms) OS_WAITT((uint)(TPM(M) + TPS(S) + TPMS(ms)))

#define OS_WAITS_A(M,S,ms) OS_WAITS((uint)(TPM(M) + TPS(S) + TPMS(ms)))

#define OS_PERIODIC_A(M,S,ms) OS_PERIODIC((uint)(TPM(M)+TPS(S)+TPMS(ms)))

#define OS_WAIT4MSG(m,t) OS_WAIT_MESSAGE(##m,##t); OS_GET_MSG(##m)

/*
***************************************************************************
*/
```

```
/*
* Other functions used internally by the RTOS :
* ---------------------------------------------
*/

void V_TaskChange (void);                 // Task swapping function
void PE_TaskChange (void);                // Task swapping function
void RTOS_Timer_Int (void);               // RTOS Scheduler ISR
void Xtra_Int (uchar task_intflag);
// Function used by ISRs other than the RTOS Scheduler

#if (!STAND_ALONE_ISR_00)
void Xtra_Int_0 (void);                   // External Interrupt 0 ISR
#endif
#if ( (TICK_TIMER != 0 ) && (!STAND_ALONE_ISR_01) )
        void Xtra_Int_1 (void);                   // Timer 0 ISR
#endif

#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2 (void);            // External Interrupt 1 ISR
#endif

#if ( (TICK_TIMER != 1 ) && (!STAND_ALONE_ISR_03) )
        void Xtra_Int_3 (void);                   // Timer 1 ISR
#endif

#if (!STAND_ALONE_ISR_04)
void Xtra_Int_4 (void);                           // Serial Port ISR
#endif
```

```
#if ( (TICK_TIMER != 2 ) && (!STAND_ALONE_ISR_05) )
void Xtra_Int_5 (void); // Interrupt 5 (Timer 2) is not available
                              // on the 8051
#endif
/*
***************************************************************************
***************************************************************************
***************************************************************************
***************************************************************************
*/
```

Parameters.h

```
/*
**************************************************************************
*                       PARAMETERS.H --- RTOS KERNEL HEADER FILE
*
* For use with MagnOS_V01.C
* Co-Operative RTOS written in C by Ing. Paul P. Debono
*                       for use with the 8051 family of microcontrollers
*
* File          : Parameters_V01.H
* Revision      : 8
* Date          : February 2006
* By            : Paul P. Debono
*
*                        B. Eng. (Hons.) Elec. Course
*                        University Of Malta
*
**************************************************************************
*/

/*
**************************************************************************
*                               RTOS USER DEFINITIONS
**************************************************************************
*/

#define STACKSIZE 0x10
               // Max size of stack for each task - no need to change
#define CPU             8032           // set to 8051 or 8032
#define TICK_TIMER 2            // Set to 0, 1 or 2 to select which timer to
                              // use as the RTOS tick timer
#define TICKTIME        50             // Length of RTOS basic tick in msec
                              // - refer to the RTOS
                              // timing definitions

#define NOOFTASKS       6        // Number of tasks used in the application program

/*
**************************************************************************
**************************************************************************
**************************************************************************
*/
```

Startup.a51

```
$NOMOD51
;------------------------------------------------------------------------
; This file is part of the C51 Compiler package
; Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
;------------------------------------------------------------------------
; STARTUP.A51: This code is executed after processor reset.
;
; To translate this file use A51 with the following invocation:
;
; A51 STARTUP.A51
;
; To link the modified STARTUP.OBJ file to your application use the following
; BL51 invocation:
;
; BL51 <your object file list>, STARTUP.OBJ <controls>
;
;------------------------------------------------------------------------
;
; User-defined Power-On Initialization of Memory
;
; With the following EQU statements the initialization of memory
; at processor reset can be defined:
;
;                              ; the absolute start-address of IDATA memory is always 0
;IDATALEN      EQU     80H     ; the length of IDATA memory in bytes.
IDATALEN       EQU     100H    ; the length of IDATA memory in bytes for
;                                the 8032 (256 bytes).
;
```

```
XDATASTART     EQU    0H     ; the absolute start-address of XDATA
; memory
XDATALEN       EQU    0H     ; the length of XDATA memory in bytes.
;
PDATASTART     EQU    0H     ; the absolute start-address of PDATA
;                              memory
PDATALEN       EQU    0H     ; the length of PDATA memory in bytes.
;
; Notes:       The IDATA space overlaps physically the DATA and BIT
;              areas of the
;              8051 CPU. At minimum the memory space occupied from the C51
;              run-time routines must be set to zero.
;------------------------------------------------------------------------
;
; Reentrant Stack Initilization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialise it:
;
; Stack Space for reentrant functions in the SMALL model.
IBPSTACK       EQU    1      ; set to 1 if small reentrant is used.
IBPSTACKTOP    EQU    0FFH+1 ; set top of stack to highest location+1.
;IBPSTACKTOP   EQU    07FH+1 ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
XBPSTACK       EQU    0      ; set to 1 if large reentrant is used.
XBPSTACKTOP    EQU    0FFFFH+1 ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the COMPACT model.
PBPSTACK       EQU    0      ; set to 1 if compact reentrant is used.
PBPSTACKTOP    EQU 0FFFFH+1  ; set top of stack to highest location+1.
;
;------------------------------------------------------------------------
;
; Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
; The following EQU statements define the xdata page used for pdata
; variables. The EQU PPAGE must conform with the PPAGE control used
; in the linker invocation.
;
PPAGEENABLE    EQU    0      ; set to 1 if pdata object are used.
;
PPAGE          EQU    0      ; define PPAGE number.
;
PPAGE_SFR      DATA   0A0H   ; SFR that supplies uppermost address byte
;              (most 8051 variants use P2 as uppermost address byte)
;
;------------------------------------------------------------------------
; Standard SFR Symbols
ACC     DATA    0E0H
B       DATA    0F0H
SP      DATA    81H
DPL     DATA    82H
DPH     DATA    83H
```

```
                NAME            ?C_STARTUP
?C_C51STARTUP   SEGMENT         CODE
?STACK S        EGMENT          IDATA

                RSEG            ?STACK
#include <parameters.h>
MAINSTACK:      DS              STACKSIZE      ; defined in parameters.h

                EXTRN CODE (?C_START)
                PUBLIC ?C_STARTUP
                PUBLIC MAINSTACK

; FLT32 or MON51 should be define in A51 TAB in Target Options
$IF (FLT32)
                CSEG    AT      8100H   ; for FLT-32
$ELSEIF (MON51)
                CSEG    AT      8000H   ; for MON-51
$ELSE
                CSEG    AT      0       ; for simulator etc
$ENDIF

?C_STARTUP:     LJMP    STARTUP1

                RSEG    ?C_C51STARTUP

STARTUP1:

IF IDATALEN <> 0
                MOV     R0,#IDATALEN - 1
                CLR     A
IDATALOOP:      MOV     @R0,A
                DJNZ    R0,IDATALOOP
ENDIF

IF XDATALEN <> 0
                MOV     DPTR,#XDATASTART
                MOV     R7,#LOW (XDATALEN)
 IF (LOW (XDATALEN)) <> 0
                MOV     R6,#(HIGH (XDATALEN)) +1
 ELSE
                MOV     R6,#HIGH (XDATALEN)
 ENDIF
                CLR     A
XDATALOOP:      MOVX    @DPTR,A
                INC     DPTR
                DJNZ    R7,XDATALOOP
                DJNZ    R6,XDATALOOP
ENDIF

IF PPAGEENABLE <> 0
                MOV     PPAGE_SFR,#PPAGE
ENDIF

IF PDATALEN <> 0
                MOV     R0,#LOW (PDATASTART)
                MOV     R7,#LOW (PDATALEN)
                CLR     A
```

```
PDATALOOP:      MOVX    @R0,A
                INC     R0
                DJNZ    R7,PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
                MOV     ?C_IBP,#LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
                MOV     ?C_XBP,#HIGH XBPSTACKTOP
                MOV     ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
                MOV     ?C_PBP,#LOW PBPSTACKTOP
ENDIF

                MOV     SP,#?STACK-1
; This code is required if you use L51_BANK.A51 with Banking Mode 4
; EXTRN CODE (?B_SWITCH0)
;               CALL ?B_SWITCH0 ; init bank mechanism to code bank 0
                LJMP ?C_START

                END
```

Appendix F Further Examples

We list here some interesting examples for the 8032 microprocessor. Some of them do not use any RTOS at all, but rely solely on interrupts.

Timer 0 in Mode 3 (split timer) and Timer 1 as a baudrate generator

The first example is a program showing how we can use Timer 0 in the split mode. This is not often found detailed in books, probably because nowadays, most of the advanced versions of the 8051 have 4 or more timers available. However, if still using the original 8051, this mode 3 would effectively increase the number of timers available.

In this example, the two timers from Timer 0 (here labeled as Timer 00 and Timer 000) both run as an 8-bit timer, generating interrupts. The main program checks if the required number of interrupts have been generated, and prints a statement accordingly.

Timer 1 is used as a baudrate generator and since Timer 0 is running in mode 3, the only way to switch on and off the timer is by changing its mode. If timer 1 is set to mode 3, it is stopped. Thus as an example, we are starting the timer only before printing and switching it off once we are done with the printing command.

```
/* TimersMode3.c */

/*
 Timer 0 runs in mode 3 mode, thus splitting it into two timers, Timer00 and Timer000:
 Timer 00 generates interrupts every 156.25us, counting 144
              hence 6400 interrupts would be equivalent to 1 second

 Timer 000 generates interrupts every 78.125us, counting 72
              hence 38400 interrupts would be equivalent to 3 seconds.

 Timer 1 is used as the baud-rate generator, switching it on and off
              by swswitching it out of and into its own mode 3.

*/
#include <reg51.h>
#include <stdio.h>

void SetUp_Timer0_M3 (void);
void SetUp_Timer1_M3 (void);
char putchar (char c);

/* Global variables */
bit T00_FLAG, T000_FLAG; // flags to indicate timer timeouts
/* ------------------------------------------------ */

/*
 * putchar : outputs character, used by the printf command
 */
```

```
char putchar (char c) {
 while (!TI);   /* wait for transmitter to be ready */
 TI = 0;
 return (SBUF = c);
}

/* ------------------------------------------ */

/* set up Timer 0 mode 3, GATE = C/T = 0 */
/* splitting into two timers, Timer00 and Timer000 */
/* Assuming 11.0592 MHz clock */

/* 156.25 microsecond overflow for TF0 (normal Timer 00) */
/* 78.125 microsecond overflow for TF1 (extra Timer 000) */
void SetUp_Timer0_M3 (void)
{
 TMOD &= 0xF0; // clear Timer 0 control bits only
 TMOD |= 0x03; // mode 3 (two split timers), GATE = C/T = 0
 TL0 = 112;    // 256 - 144 ==> 156.25us for normal Timer 00
 TH0 = 184;    // 256 - 72 ==> 78.125us for extra Timer 000
 TR0 = 1;              // Timer 00 ON
 TR1 = 1;      // Timer 000 ON
 ET0 = 1;              // Enable TF0 interrupt, from Timer 00 overflow
 ET1 = 1;              // Enable TF1 interrupt, from Timer 000 overflow
}

/* ------------------------------------------ */

/* Set up timer 1 in mode 2, 8-auto re-load, GATE = C/T = 0 */
/* for 57600 baudrate generator */
/* Assuming 11.0592 MHz clock */

/* Since Timer 0 is in mode 3, then Timer 1 will be switched on and off
   by setting it to mode 2 (on) or mode 3 (off) in the application program */

/* Set also the UART */
void SetUp_Timer1_M3 (void)
{
 TMOD &= 0x0F;         // clear timer 1 control bits only (momentarily set T1 to mode 0)
 TMOD |= 0x30;         // set initially to mode 3, i.e. timer off
 TH1 = TL1 = 0xFF;     // set for 28800 or 57600 baudrate (reload value in TH1)
 PCON |= 0X80;         // SMOD = 1 so as to double the baudrate
 SCON = 0X52;          // 8-bit UART, variable baudrate, receiver disabled ,
                       // transmitter ready TI = 1
}

/* ------------------------------------------ */

/* -------------------------------------------------- */

void TF0_ISR (void) interrupt 1 using 1
{
static data unsigned int TF0_OVF;     // counts TF0 overflows
 TF0_OVF++;
 TL0 = 112;
 if (TF0_OVF == 6400)
        {
```

```
                TF0_OVF = 0;
                T00_FLAG = 1;
        }
}
/* ------------------------------------------------ */
void TF1_ISR (void) interrupt 3 using 2
{
static data unsigned int TF1_OVF;      // counts TF1 overflows
 TF1_OVF++;
 TH0 = 184;
 if (TF1_OVF == 38400)
        {
                TF1_OVF = 0;
                T000_FLAG = 1;
        }
}

/* ------------------------------------------------ */

/* ------------------------------------------------ */

/* Main program */
void main(void)
{
SetUp_Timer0_M3 ();             // Timer 0 mode 3 - split timer
SetUp_Timer1_M3 ();             // Timer 1 (off) mode 3,
                                // 8-bit auto reload value as a baudrate generator
                                // initially set in mode 3, not running.
```

```
EA = 1;

while(1)
 {
// Timer 1 is switched on and off just to show that we can still control it.
// It is switched only for use as the baud rate generator before the printf command
   if (T00_FLAG == 1)
        {
         T00_FLAG = 0;
         TMOD = 0x23;  // set Timer 1 to mode 2, start it as the baudrate generator
                              // leaving Timer 0 set to mode 3
                              // This method is used instead of:
                              //
                              // TMOD &= 0x0F;       // clear timer 1 control bits only
                                                     // (momentarily set T1 to mode 0)
                              // TMOD |= 0x20;       // set to mode 2, i.e. Timer 1 on
                              // TH1 = 0xFF;         // set reload value
                              //
                              // which would have placed Timer 1 momentarily in mode 0
                              // and thus modifying the reload value held in TH1
                              // (hence the baudrate) before setting it to mode 2
                              // Hence the need to set the reload value in TH1 every time.
                              // Thus TMOD = 0x23 is much quicker and neater this time!

         printf ("Timer 00 timeout every 1 second\n");
         TMOD = 0x33;         // set Timer 1 to mode 3 to stop the baudrate generator
                              // leaving Timer 0 set to mode 3
 }
 if (T000_FLAG == 1)
        {
         T000_FLAG = 0;
         TMOD = 0x23;         // set Timer 1 to mode 2, start it as the baudrate generator
                              // leaving Timer 0 set to mode 3
         printf ("Timer 000 timeout every 3 seconds\n");
         TMOD = 0x33;         // set Timer 1 to mode 3 to stop the baudrate generator
                              // leaving timer 0 set to mode 3
 }
 }
}
```

UART using Timer 2 as the baud rate generator

```
/* SerialTimer2.c */

/*
 Timer 0 runs in mode 2 mode (8-bit auto reload)
 Timer 0 generates interrupts every 156.25us, counting 144
              hence 6400 interrupts would be equivalent to 1 second

 Timer 1 runs in mode 2 mode (8-bit auto reload)
 Timer 1 generates interrupts every 78.125us, counting 72
              hence 38400 interrupts would be equivalent to 3 seconds.

 Timer 2 is used as the baud-rate generator, switching it on and off
              just when printing is needed (just for demo).
*/
#include <reg52.h>      // use 8052/8032 SFR registers
#include <stdio.h>

void SetUp_Timer0_M2 (void);
void SetUp_Timer1_M2 (void);
void SetUp_Timer2_Serial (void);
char putchar (char c);

/* Global variables */
bit T0_FLAG, T1_FLAG; // flags to indicate timeout

/* ------------------------------------------------ */

/*
 * putchar : outputs character, used by the printf command
 */
char putchar (char c) {
 while (!TI);   /* wait for transmitter to be ready */
 TI = 0;
 return (SBUF = c);
}

/* ------------------------------------------ */

/* Set up Timer 0 mode 2, 8-bit timer auto reload, GATE = C/T = 0 */
/* Assuming 11.0592 MHz clock */

/* 156.25 microsecond overflow for TF0 */
void SetUp_Timer0_M2 (void)
{
 TMOD &= 0xF0; // clear timer 0 control bits only
 TMOD |= 0x02; // mode 2 (8-bit reload), GATE = C/T = 0
 TL0 = 112;    // 256 - 144 = 112 ==> 156.25us for Timer 0
 TH0 = 112;    // reload value in TH0
 TF0 = 0;              // Clear overflow flag
 TR0 = 1;              // Timer 0 ON
 ET0 = 1;              // Enable TF0 interrupt, from Timer 0 overflows
}

/* ------------------------------------------ */
```

```
/* Set up Timer 1 in mode 2, 8-bit auto reload, GATE = C/T = 0 */
/* Assuming 11.0592 MHz clock */

/* 78.125 microsecond overflow for TF1 */

void SetUp_Timer1_M2 (void)
{
 TMOD &= 0x0F;          // clear Timer 1 control bits only (momentarily set T1 to mode 0)
 TMOD |= 0x20;          // set to mode 2
 TH1 = TL1 = 184;       // 256 - 72 = 184 ==> 78.125us for Timer 1 (reload value in TH1)
 TF1 = 0;               // Clear overflow flag
 TR1 = 1;               // Timer 1 ON
 ET1 = 1;               // Enable TF1 interrupt, from Timer 1 overflows
}

/* ------------------------------------------ */

/* Set up Timer 2 in mode 2, 16-bit auto reload, GATE = C/T = 0 */
/* for 345600 baudrate generator */
/* Assuming 11.0592 MHz clock */

/* Serial Port is also set to use Timer 2 as the baud rate generator for Tx and Rx */

void SetUp_Timer2_Serial (void)
{
 C_T2 = 0;                      // Timer 2 in timer mode
 TH2 = TL2 = 0xFF;              // set for 345600 baudrate (reload value in TH1)
 RCAP2H = RCAP2L = 0xFF;        // reload values
 RCLK = TCLK = 1;               // use Timer 2 as the baud rate generator for Rx and Tx
 SCON = 0X52;           // 8-bit UART, variable baudrate, receiver disabled ,
transmitter ready TI = 1
}

/* ------------------------------------------ */

/* ------------------------------------------------ */

void TF0_ISR (void) interrupt 1       using 1
{
static data unsigned int TF0_OVF;      // counts TF0 overflows
 TF0_OVF++;
 if (TF0_OVF == 6400)
        {
               TF0_OVF = 0;
               T0_FLAG = 1;
        }
}

/* ------------------------------------------------ */

void TF1_ISR (void) interrupt 3       using 2
{
static data unsigned int TF1_OVF;      // counts TF1 overflows
 TF1_OVF++;
 if (TF1_OVF == 38400)
        {
               TF1_OVF = 0;
               T1_FLAG = 1;
        }
```

```
}

/* ------------------------------------------------ */

/* ------------------------------------------------ */

/* Main program */
void main(void)
{
SetUp_Timer0_M2 ();            // Timer 0 mode 2
SetUp_Timer1_M2 ();            // Timer 1 mode 2
SetUp_Timer2_Serial ();        // Timer 2 and serial port set up.
                               // Timer 2 as the 345600 baud rate generator.
EA = 1;                        // enable the 8051 to accept interrupts

while(1)
 {
// Timer 2 is switched on and off just to show that we can still control it.
// It is switched on only for use as the baud rate generator before the printf command
// It is usually left running throught the whole program
// (unless you want to reduce battery consumption!)

 if (T0_FLAG == 1)
        {
         T0_FLAG = 0;
         TR2 = 1;              // start Timer 2 as the baudrate generator
         printf ("Timer 0 timeout every 1 second\n");
         TR2 = 0;              // stop the baudrate generator
 }

 if (T1_FLAG == 1)
        {
         T1_FLAG = 0;
         TR2 = 1;              // start Timer 2 as the baudrate generator
         printf ("Timer 1 timeout every 3 seconds\n");
         TR2 = 0;              // stop the baudrate generator
 }
 }
}
```

Serial routine with full XON/XOFF capability

The next example program here is a serial routine with full XON/XOFF handshaking capability. This was adapted from a program by Sasha Jevtic (sjevtic@ece.northwestern.edu).

It uses two circular buffers, one to hold the received bytes and another separate one for the bytes to transmitted. The principle behind this XON/XOFF protocol is explained below, and uses two special characters to control the transmission flow of data.

CTRL-S which is 17 decimal or 11 hexadecimal, used to stop the transmission.
CTRL-Q which is 19 decimal or 13 hexadecimal, used to continue (resume) the transmission.

For the RECEIVER side, a character reception from some external source, causes an RI interrupt. The RI ISR routine therefore handles any characters which are received. If there is space in the RX buffer, it simply stores it there but if the buffer approached the limit, it will store that character and causes the TX to send an XOFF character to the external source (which should also be running under an XON/XOFF protocol) so that the external source pauses its transmission. Data from the RX buffer would be handled by the main program which should ensure that the buffers are emptied regularly of any data. Once the RX buffer is emptied, an XON character would be sent to the external source so that it can resume with its transmission.

If an XOFF character is received, this would cause the TX to be disabled immediately (since this would imply that the external source RX buffer is full). It would be re-enabled once an XON character is received.

For the TRANSMITTER side, data from the main program would be written to the buffer. As soon as there is some data in the TX buffer, the TI ISR would be triggered to start the transmission. Since the TI is itself set once a character has been transmitted, this would ensure that the TX buffer is always emptied once the transmission is started (unless it is stopped by an XOFF character from the external device). Therefore the TI ISR handles the transmission from the buffer on to some external device. The main program 'printing' routine would place characters directly in the TX buffer for transmission, if there is space. If the TX buffer is full, it would simply wait for room in the buffer, since the transmission would be taking place under interrupt control.

Since this protocol uses XON/XOFF as special characters, it is generally used for text data (which does not have the XON/XOFF characters) and it cannot be used as it is with any random data since they might have these XON/XOFF characters as part of the data itself.

Further details can be found in the remarks within the program listings.

```
/*
** XONXOFF.H
*/

#ifndef SERIAL_H
#define SERIAL_H

#include <reg52.h>

// Some commonly used non-printing ASCII codes.

#define CTRL_C          0x03
#define CTRL_Q          'q'     // for testing
//#define CTRL_Q        0x11
#define CTRL_S          's'     // for testing
 //#define CTRL_Q       0x13
#define DEL             0x7F
#define BACKSPACE       0x08
#define CR              0x0D
#define LF              0x0A
#define BELL            0x07
```

```
#ifdef SMALLBUFFER
 #define TXLEN          128
 #define RXLEN          16
 #define RXThreshHold   8       /* Rx Threshhold for sending XOFF */
#else
 #define TXLEN          1024
 #define RXLEN          1024
 #define RXThreshHold   512     /* Rx Threshhold for sending XOFF */
#endif
void putbuf(char c);
char putchar(char c);
char _getkey(void);
void init_serial(unsigned int br);

#endif  // SERIAL_H
-----------------------*******************-------------------------------

** XonXoff.c
**
** Modified by Paul Debono (2007) from
** RTMS 2.0 I/O: serial.c v1.0.0 (16/03/05)
** By Sasha Jevtic (sjevtic@ece.northwestern.edu)
**
** Implements an interrupt controlled serial port interface
**
** TRANSMITS,RECEIVES AND REACTS TO XON/XOFF characters for flow control
**
*/

#include "XonXoff.h"

typedef enum {NORMAL, FULL, DRAINING, EMPTY} RECV_STATE_T;

/*The way the pointers work is as follows:
 Pointers are always incremented by 1, modulus LEN
 That is if LEN = 32, pointers range form 0 to 31

 Tx Buffer:
 Data for transmission is placed in the buffer in the location pointed to by 't_in'.
 Data is passed on to SBUF for transmission via UART using pointer 't_out'.
 In each case, the pointer is moved AFTER reading or writing from/to buffer,
 Thus   t_out points to the next character to be sent to SBUF.
        t_in points to the location where the next character will be placed
                in Tx buffer for future transmission

 If AFTER incrementing 't_out',
        't_in' and 't_out' point to the same location,
               then the buffer is empty.

 If AFTER incrementing 't_in',
        't_in' = 't_out' point to the same location,
               then the buffer is full

 Rx Buffer:
 Data is received in the buffer in the location pointed to by 'r_in'
 Data is read from the buffer using pointer 'r_out'
 In each case, the pointer is moved AFTER reading or writing from/to buffer
```

```
Thus r_out points to the next character to be read.
       r_in points to the location where the next character will be placed
                when it is received.

If AFTER incrementing 'r_out',
       'r_in' and 'r_out' point to the same location,
       then the buffer is empty.

If AFTER incrementing 'r_in',
       'r_in' = 'r_out' point to the same location,
               then the buffer is full

SERIAL INTERRUPTS ARE DISABLED WHEN HANDLING BUFFERS

Meanings of receiver states (and related policies) are as follows:
- NORMAL:     Receive buffer is empty or filling, but is not full.
                      Reception should proceed normally.

- FULL:       Receive buffer has been marked full. We need to send an
                      XOFF character so that the sender allows us to relieve our
                              buffer.
- EMPTYING:   Receive buffer is full or emptying, but is not empty.
                      An XOFF character has already been sent, so reception should
                              be suspended until the buffer is empty. If we permitted
                              reception prior to completely emptying the buffer, we would
                              put ourselves in a situation where it is very likely that the
                              buffer would soon fill up again. This would be inefficient,
                              as we wish to keep the XON/XOFF : data byte ratio very low.
              This hysteresis helps to achieve that goal.
```

```
 - EMPTY:      Receive buffer has been marked empty. We need to send an XON
               character so that the sender begins sending us data again.

 It is worth noting that for Windows communication, a significant receive
 buffer is required. Furthermore, the threshhold level at which XOFF is sent
 must be significantly below the buffer's capacity. This is required for a
 couple of reasons.
 First, there might already be an incoming byte working its way through the
 UART at the time that XOFF is sent. Even assuming an instantaneous response
 from the DTE to our XOFF, if space does not become available in the buffer
 prior to the firing of the receive interrupt for this byte, it will be lost.

 Secondly, and moreover, Windows seems to be very slow in responding to our
 XOFF; that is, it continues to send data for a significant period before
 honoring our stop request. So, we need space to buffer that incoming data.
 Received data integrity is a priority.
*/

#ifdef SMALLBUFFER     // use IDATA, character size pointers
 unsigned char data t_out;    /* transmission buffer start index   */
 unsigned char data t_in;     /* transmission buffer end index     */
 char idata TxBuffer[TXLEN];  /* storage for transmission buffer   */

 // It seems we need 128 bytes of buffer to run at 57600 or 115200 kbps.
 // when receiving characters from a PC (Windows)
 // These values depend on sender reaction time (to XOFF) and on the baudrate

 unsigned char data r_out;    /* receiving buffer start index      */
 unsigned char data r_in;     /* receiving buffer end index        */
 char idata RxBuffer[RXLEN];  /* storage for receiving buffer      */

#else // LARGEBUFFER - use XDATA, integer size pointers

 unsigned int data t_out;     /* transmission buffer start index */
 unsigned int data t_in;      /* transmission buffer end index */
 char xdata TxBuffer[TXLEN];  /* storage for transmission buffer */

 // It seems we need 128 bytes of buffer to run at 57600 or 115200 kbps.
 // when receiving characters from a PC (Windows)
 // These values depend on sender reaction time (to XOFF) and on the baudrate

 unsigned int data r_out;     /* receiving buffer start index      */
 unsigned int data r_in;      /* receiving buffer end index        */
 char xdata RxBuffer[RXLEN];  /* storage for receiving buffer      */
#endif // buffer size

bit TxBfull;                  /* flag: marks transmit buffer full   */
bit TxActive;                 /* flag: marks transmitter active     */
bit TxStop;                   /* flag: marks XOFF character         */
bit TxBusy;                   /* flag: marks transmitter busy       */
bit TxBempty;                 /* flag: marks transmit buffer empty  */
bit RxBempty;                 /* flag: marks receive buffer empty   */
bit RxBfull;                  /* flag: marks receive buffer full    */
RECV_STATE_T recvstate;       /* receiver state, for flow control   */
enum {FALSE,TRUE} CONDITION_T;
```

```
#ifdef SMALLBUFFER
unsigned char RxBufUsed()
{
unsigned char size;
 if(r_in >= r_out)
 size = (r_in - r_out);
 else
 size = (RXLEN-(r_out - r_in));
 return(size);
}

bit RxBufOverTHLD()
{
 return(RxBufUsed()>RXThreshHold ? TRUE : FALSE);
}

#else
unsigned int RxBufUsed()
{
unsigned int size;
 if(r_in >= r_out)
       size = (r_in - r_out);
 else
       size = (RXLEN-(r_out - r_in));
 return(size);
}

bit RxBufOverTHLD()
{
 return(RxBufUsed()>RXThreshHold ? TRUE : FALSE);
}

#endif

/*****************************************************************************/
/*      putbuf: write a character to SBUF or transmission buffer */
/*****************************************************************************/

void putbuf(char c)
{
 if (!TxBfull) {                     /* transmit only if buffer not full   */
/*
 Note that if buffer is full, waiting is handled by putchar routine
 which calls putbuf.
*/
 ES = 0;                             /* disable serial interrupt           */

 if (!TxActive && !TxStop) {         /* if transmitter not active:         */
       TxActive = 1;                 /* transfer the first character direct */
       SBUF = c;                     /* to SBUF to start transmission      */
 }
 else {                              /* otherwise: */
       TxBuffer[t_in] = c;           /* transfer char to transmit buffer   */
        t_in = ++t_in & (TXLEN-1);   /* at the same time incrementing      */
        TxBempty = 0;                /* the pointer in circular fashion    */
       if (t_in == t_out) {
```

```
        TxBfull = 1;                  /* set flag if buffer is full          */
    }
  }
        ES = 1;                       /* enable serial interrupt             */
  }
}

/***************************************************************************/
/*      putchar:                                                *\/
/*      Places character in Tx buffer if there is space         */
/*      otherwise wait (space created by serial ISR)            */
/***************************************************************************/

char putchar(char c)
{
 if (c == '\n') {               /* expand new line character:         */
                                /* can omit if not needed             */
 while (TxBfull) {;}            /* wait for transmission buffer space */
                                /* which is cleared by the serail ISR */
 putbuf (0x0D);                 /* send CR before LF for <new line>   */
 }

 while(TxBfull) {;} /* wait for transmission buffer space */

 putbuf(c); /* send character */
 return(c); /* return character: ANSI requirement */
}

/***************************************************************************/
```

```
/*      _getkey:                                                        */
/*      Read data from Rx buffer (using scanf say)                      */
/*       Waits if no character, until one is received via serial ISR    */
/************************************************************************/

char _getkey(void)
{
 unsigned char data c;

 while (RxBempty) {;}
 /*
 wait for a character in Rx buffer,
 in the mean time, any RI interrupt will fill buffer,
        and therefore RX buffer will no longer remain empty

 */
 ES = 0;
 c = (RxBuffer[r_out]);              /* take next char from buffer        */
 r_out = ++r_out & (RXLEN-1);
 if(r_out==r_in) RxBempty = 1;
 if ((recvstate == DRAINING) &&      /* if RX was full and now emptying... */
 (RxBempty)) {                       /* ...and RX actually has emptied    */
 recvstate = EMPTY;                  /* prepare to send XON               */
 TI = 1;                             /* force TI so as to send XON        */
 }
 ES = 1;
 return(c);
}

/************************************************************************/
/*             serial: serial receiver / transmitter interrupt */
/************************************************************************/

serial () interrupt 4 using 1        /* use registerbank 1 for interrupt  */
{
 unsigned char c;
 bit start_trans = 0;
 if (RI) {                           /* if receiver interrupt             */
 c = SBUF;                           /* read character                    */
 RI = 0;                             /* clear interrupt request flag      */

 switch (c) {                        /* process character                 */

 case CTRL_S:                        /* XOFF                              */
 TxStop = 1;                         /* if Control+S stop transmission    */
 break;

 case CTRL_Q:                         /* XON */
        start_trans = TxStop;        /* if Control+Q start transmission   */
        TxStop = 0;
        break;

 default:                            /* put all other characters into RxBuffer */

 if (!RxBfull) {
        RxBuffer[r_in] = c;
```

```
                r_in = ++r_in & (RXLEN-1); /* check if over thresh hold is done*/
                RxBempty = 0;                 /* below instead of just checking if full */
 }

               /* check if RX above XOFF threshhold */
       if (RxBufOverTHLD())
       {
       if (recvstate == NORMAL) {     /* prevent "oscillations"             */
               recvstate = FULL;             /* prepare to send XOFF                */
                       RxBfull = 1;
       }
 }
 break;
 }

 if (TxBusy) return;   /* do not send anything until transmitter free      */
                       /* It will return to the ISR when TI=1 after tx      */
 }                                                    // end of RI

 if (TI ||                    /* if transmitter interrupt                   */
       start_trans ||         /* or we received an XON and must start       */
       (recvstate == FULL) || /* or we need to send an XOFF                 */
       (recvstate == EMPTY)) {/* or we need to send an XON                  */

 if (TI) {
       TxBusy = 0;            /* TX interrupt means not busy anymore        */
       TI = 0;                /* clear interrupt request flag               */
       }

 if (recvstate == FULL) {     /* we need to send an XOFF                    */
       TxBusy = 1;
       SBUF = CTRL_S;         /* send XOFF command to other sender          */
       recvstate = DRAINING; /* slowly wait for RX buffer to drain          */
 }

 else if (recvstate == EMPTY) { /* we need to send an XON                   */
       TxBusy = 1;
       SBUF = CTRL_Q;                         /* send XON to sender         */
       recvstate = NORMAL;    /* we are back in business, receiving         */
 }

 else if (!TxBempty) {        /* if more characters in buffer and           */

 if ((!TxStop) &&             /* if not received Control+S (XOFF)           */
       (recvstate == NORMAL)) { /* and receive buffer isn't overwhelmed     */
       TxBusy = 1;
       SBUF = TxBuffer[t_out]; /* transmit character                        */
               t_out = ++t_out & (TXLEN-1);
               if (t_out==t_in) TxBempty =1;
 TxBfull = 0; /* clear 'TxBfull' flag */
 }

 }
 else TxActive = 0;           /* if all transmitted sender not active      */
 }
}
```

```
void init_serial(unsigned int baudrate)
{
SCON = 0x50;               /* Setup serial port control register */
                           /* Mode 1: 8-bit uart var. baud rate */
                           /* REN: enable receiver, TI=0 */

PCON &= 0x7F;              /* Clear SMOD bit in power ctrl reg (no double baudrate) */

TMOD &= 0x0F;              /* Setup timer/counter mode register */
                           /* Clear M1 and M0 for timer 1 */
TMOD |= 0x20;              /* Set M1 for 8-bit auto-reload timer 1 */

RCLK = 0;                  /* USE TIMER 1 FOR RECEIVE BAUD RATE (8032 only) */
TCLK = 0;                  /* USE TIMER 1 FOR TRANSMIT BAUD RATE (8032 only) */

switch (baudrate) {
 case 600:
        TH1 = TL1 = 0xD0;
                break;
 case 1200:
        TH1 = TL1 = 0xE8;
                break;
 case 2400:
        TH1 = TL1 = 0xF4;
                break;
 case 4800:
        TH1 = TL1 = 0xFA;
                break;
```

```
 case 9600:
        TH1 = TL1 = 0xFD;
                break;
 case 19200:
        TH1 = TL1 = 0xFD;
                PCON |= 0x80; /* double baudrate */
                break;
 case 57600:
        TH1 = TL1 = 0xFF;
                PCON |= 0x80; /* double baudrate */
                break;
 }

 // Make sure we start in a clean state.

 TxBempty = 1;
 TxBfull = 0;
 TxActive = 0;
 TxStop = 0;
 TxBusy = 0;

 RxBempty = 1;
 RxBfull = 0;

 recvstate = NORMAL;
 t_out = 0;
 t_in = 0;

 r_out = 0;
 r_in = 0;

 TI = RI = 0;
 TR1 = 1; /* Start timer 1 for baud rate generation */
 ES = 1;                                 /* enable serial interrupts */
 EA = 1;                                 /* enable global interrupts */
}
/*--------------------------------------------------------------------------------
--------------------------------------------------------------------------------*/
```

Appendix G 8086 PaulOS RTOS

8086 PaulOS RTOS Demo Program

Sometimes, we had encountered instances where students wanted to use some 8086-based development board as their embedded system. An 8086 version of the basic PaulOS RTOS was therefore developed and is being given here just for the benefit of those who are keen on the 8086 processor. This is in fact just a demonstration version of the PaulOS RTOS program, written for the Intel 8086/8088 micro-processor. It can be compiled on the latest PCs using any good 8086 assembler (such as MASM) to produce the .COM file which can then be executed directly.

It is composed of four modules:

- The main module CLOCK1.ASM, which basically contains the application program and 'includes' the other .INC files
- The RTOS 'include' file OSTICKHD.INC – header
- The RTOS 'include' file OSTICKMD.INC – middle
- The RTOS 'include' file OSTICKFT.INC – footer

The .INC files are the actual PaulOS RTOS program, written in assembly language for the 8086.

The main program is a simple clock, displaying time in HH:MM:SS format. It uses three tasks, one task for the hour, another task for the minute and the last task for the second. The MINUTE and HOUR tasks are always waiting for a signal while the SECOND task is simply waiting for a time delay of 1 second. Once 60 seconds have passed, it sends a signal to the MINUTE task. Similarly once 60 minutes have passed, the MINUTE task would send a signal to the HOUR task.

```
; CLOCK1.ASM
;       Using BIOS INT to get time ticks since midnight
;
;       Used to generate rtos ticks
;
;
;       Written by Paul P. Debono ( JAN 2012 )
;
;=========================================================

; EQUATES

; APPLICATION SETTINGS
NUM_OF_TASKS   EQU 3
IDLE_TASK      EQU NUM_OF_TASKS
include ostickhd.inc

;=============================================================
;=============================================================

CODESEG         SEGMENT
ASSUME          CS:CODESEG, DS:CODESEG, SS:CODESEG ; advices MASM
                                       ; which segments to use

ORG 100H ; start at offset 100H FOR COM FILE

START:          MOV AX, CS
                MOV DS,AX
                MOV SS,AX
                MOV SP, OFFSET TOP_OF_STACK
                MOV WORD PTR [MY_CODE_SEG], CS ; STORAGE FOR CS, WHENEVER NEEDED
                        JMP MAIN

;=============================================================

;=============================================================

include ostickmd.inc

;=============================================================
;=============================================================

;=============================================================
;=============================================================
; START OF MAIN TEST (APPLICATION) PROGRAM
;=============================================================
;=============================================================
```

```
MAIN:
; PARAMETERS FOR OS_CREATE ARE THE TASK NUMBER AND THE TASK ROUTINE NAME
        OS_CREATE_TASK 0, SEC
        OS_CREATE_TASK 1, MIN
        OS_CREATE_TASK 2, HR

        MOV AH, 9 ; PRINT DOLLAR TERMINATED STRING
        MOV DX, OFFSET MESSAGE01
        INT 21H

TOP:    OS_RTOS_CHECK_TICK

        MOV AH,01H ; CHECK FOR ANY KEY PRESS TO HALT PROGRAM
        INT 16H ; BUT DO NOT WAIT FOR THE KEYPRESS
        JZ TOP
        CMP AL, 'X'
        JE HALT
        CMP AL, 'x'
        JE HALT
        JMP TOP ; PROGRAM LOOPS HERE

HALT:
        MOV AH, 4CH ; EXIT TO OS
        INT 21H


; ====================================================
;
;               TASKS
;
;       ALL TASKS MUST BE ENDLESS LOOPS
;
;

SEC PROC NEAR
                OS_PERIODIC 1000/TICKTIME_MS ; PERIOD SET FOR 1s
SEC_TASK:
                OS_WAIT_PERIOD
                CMP WORD PTR [SECONDS], 3935H ;59 ASCII INVERTED
                JE SKIP1S
                CMP BYTE PTR [SECONDS + 1], '9'
                JE SKIP2S
                INC BYTE PTR [SECONDS + 1]
                CALL DISPLAY_CLOCK
                JMP SEC_TASK
SKIP2S:         MOV BYTE PTR [SECONDS + 1], '0'
                INC BYTE PTR [SECONDS]
                CALL DISPLAY_CLOCK
                JMP SEC_TASK
SKIP1S:
                MOV WORD PTR [SECONDS], 3030H
                OS_SIGNAL_TASK 1
                JMP SEC_TASK

SEC ENDP
```

```
MIN PROC NEAR
MIN_TASK:
               OS_WAITS       ; WAIT FOR 'END OF MINUTE' SIGNAL
               CMP WORD PTR [MINUTES], 3935H
               JE SKIP1M
               CMP BYTE PTR [MINUTES + 1], '9'
               JE SKIP2M
               INC BYTE PTR [MINUTES + 1]
               CALL DISPLAY_CLOCK
               JMP MIN_TASK
SKIP2M:        MOV BYTE PTR [MINUTES + 1], '0'
               INC BYTE PTR [MINUTES]
               CALL DISPLAY_CLOCK
               JMP MIN_TASK
SKIP1M:
               MOV WORD PTR [MINUTES], 3030H
               OS_SIGNAL_TASK 2
               JMP MIN_TASK
MIN ENDP

HR PROC NEAR
HR_TASK:
               OS_WAITS ; WAIT FOR 'END OF HOUR' SIGNAL
               CMP WORD PTR [HOURS], 3332H ; 23 ASCII INVERTED
               JE SKIP1H
               CMP BYTE PTR [HOURS + 1], '9'
               JE SKIP2H
               INC BYTE PTR [HOURS + 1]
```

```
                CALL DISPLAY_CLOCK
                JMP HR_TASK

SKIP2H:         MOV BYTE PTR [HOURS + 1], '0'
                INC BYTE PTR [HOURS]
                CALL DISPLAY_CLOCK
                JMP HR_TASK
SKIP1H:
                MOV WORD PTR [HOURS], 3030H
                CALL DISPLAY_CLOCK
                JMP HR_TASK
HR ENDP
;*******************************
; THIS ROUTINE DISPLAYS THE CLOCK
;

DISPLAY_CLOCK:
        PUSH    DX

        MOV     AH, 9 ; PRINT DOLLAR TERMINATED STRING
        MOV     DX, OFFSET FRONT_SPACE
        INT     21H

        POP     DX
RET
;*******************************

;start of our data area (if neeeded)
;
; APPLICATION DATA AREA
;
; variable to store original
; value of SI register.

FRONT_SPACE     DB " "
HOURS           DB "23"
COLON1          DB ':'
MINUTES         DB "58"
COLON2          DB ':'
SECONDS         DB "40",13,'$'
MESSAGE01       DB 13,10,10," RTOS CLOCK DEMO PROGRAMME", 13,10
                DB " (PRESS X OR x TO EXIT)", 13,10,10,'$'
;
;
include ostickft.inc

TOP_OF_STACK DB 0
;end of our data area

CODESEG ENDS
END START
;==================================================
;========================================================
; OSTICKHD.INC
;
;               Written by Paul P. Debono ( JAN 2012 )
;
;========================================================
```

```
; EQUATES

; RTOS SETTINGS
TICKTIME_MS           EQU 55 ; APPROXIMATELY 55ms TICKTIME
STACK_SIZE            EQU 40
PARAM_SIZE            EQU 50 ; 10 + STACK_SIZE
PARAM_SIZE_REDUCED    EQU 10 ; USED IF A LARGE NUMBER OF TAKS ARE BEING USED
                       ; SEE END OF FILE, WHEN DECLARING TASK_PARAM

; RTOS PARAMETERS OFFSETS, FOR EACH TASK
SIGNAL_FLAG           EQU 0 ; DB 0
INT_NUM               EQU 1 ; DB 0
SP_STORE              EQU 2 ; DW 0
TIME_OUT              EQU 4 ; DW 0
PERIOD_CURRENT        EQU 6 ; DW 0
PERIOD_RELOAD         EQU 8 ; DW 0
STACK_AREA            EQU 49 ; PARAM_SIZE - 1
; points to top of stack

; EQUATES

;============================================================
;============================================================

; MACROS USED BY THE RTOS
REG_PUSHES EQU 7 ; NUMBER OF PUSHED REGISTERS IN PUSH_INT_REGS

PUSH_ALL_REGS MACRO
               PUSHF
               PUSH AX
               PUSH BX
               PUSH CX
               PUSH DX
               PUSH BP
               PUSH DI
               PUSH SI
       ENDM

POP_ALL_REGS MACRO
               POP SI
               POP DI
               POP BP
               POP DX
               POP CX
               POP BX
               POP AX
               POPF
       ENDM

PUSH_INT_REGS MACRO
               PUSH AX
               PUSH BX
               PUSH CX
               PUSH DX
               PUSH BP
               PUSH DI
               PUSH SI
       ENDM
```

```
POP_INT_REGS MACRO
               POP SI
               POP DI
               POP BP
               POP DX
               POP CX
               POP BX
               POP AX
        ENDM

OS_CREATE_TASK MACRO TASKNUM, TASKNAME
               MOV DX, OFFSET TASKNAME
               MOV AL,TASKNUM
               CALL CREATE_TASK
        ENDM
;
;
; COMMANDS WHICH CAUSE A TAK CHANGE NEED TO HAVE TWO ADDITIONAL
; REGISTER PUSHES, SINCE A TAKS CHANGE MAY ALSO BE CALLED BY THE
; TICK TIME ISR.
; HENCE WE NEED TO SIMULATE AN INTERRUPT CALL WHERE NEEDED
;
;
OS_WAITT MACRO TIMETICKS
               MOV DX, TIMETICKS
               PUSHF          ; USED TO SIMULATE AN INTERRUPT CALL
               PUSH CS        ; USED TO CALL RTOS ROUTINES WHICH
               CALL WAITT     ; CAUSE A TASK CHANGE
        ENDM

OS_PERIODIC MACRO TIMETICKS
               MOV DX, TIMETICKS
               CALL PERIODIC
        ENDM

OS_WAIT_PERIOD MACRO
               PUSHF ; USED TO SIMULATE AN INTERRUPT CALL
               PUSH CS ; USED TO CALL RTOS ROUTINES WHICH
               CALL WAIT_PERIOD ; CAUSE A TASK CHANGE
        ENDM

OS_WAITS MACRO
               PUSHF ; USED TO SIMULATE AN INTERRUPT CALL
               PUSH CS ; USED TO CALL RTOS ROUTINES WHICH
               CALL WAITS ; CAUSE A TASK CHANGE
        ENDM

OS_SIGNAL_TASK MACRO TASKNUM
               MOV AL, TASKNUM
               CALL SIGNAL_TASK
        ENDM

OS_RTOS_CHECK_TICK MACRO
; GET the new number of ticks since midnight.
; if there was a new tick, then the RTOS Tick TImer ISR
```

```
; routine must be called.

               MOV AH, 0
               INT 1AH
               CMP CS:WORD PTR [OldTick], DX ; compare with old
               JE NOCHANGE
               MOV CS:WORD PTR [OldTick], DX ; and save it
               PUSHF
               PUSH CS
               CALL OS_TICK_TIMER    ; call ticktimer ISR,
                                     ; simulating an interrupt call
NOCHANGE: NOP
        ENDM

;============================================================
;============================================================

;============================================================
; OSTICKMD.INC
;       Written by Paul P. Debono ( JAN 2012 )
;
;============================================================

;============================================================
;============================================================

; RTOS TICK TIMER INTERRUPT SERVICE ROUTINE (ISR)
;
; THIS IS THE MAIN RTOS ISR ROUTINE WHICH COMES INTO PLAY
; AT EVERY TICK TIME, AND SCHEDULES ANY TASK CHANGES REQUIRED.
;
; A NORMAL TICK TIME INTERVAL WOULD BE EVERY 1 MILLISECOND.
;
; IT DECREMENTS WAITING TICK TIMES FOR EACH TASK, IF WAITING
; AND FORCES A TASK CHANGE IF REQUIRED
; BY CHANGING THE STACK CONTENTS.

OS_TICK_TIMER PROC NEAR
                      PUSH_INT_REGS

; FIRST CHECK THE PERIODIC INTERVAL TASKS
               MOV CX,NUM_OF_TASKS
               MOV BX,OFFSET TASK_PARAM
               MOV DL,PARAM_SIZE
CHECK0:        MOV AL,CL      ; TASK NUMBERS RANGE = ( 0 TO NUM_OF_TASKS - 1)
               DEC AL         ; AL NOW CONTAINS THE HIGHEST TASK NUMBER
               MUL DL
               MOV SI,AX ; SI CONTAINS OFFSET IN TASK RECORD
               CMP WORD PTR [BX + SI + PERIOD_RELOAD],0
               JZ NEXT_CHECK0 ; SKIP IF TASK IS NOT WAITING PERIODICALLY
               DEC WORD PTR [BX + SI + PERIOD_CURRENT] ; DECREMENT TIME
               JNZ NEXT_CHECK0

; IF PERIODIC TIME OUT HAS FINISHED,
; FIRST RELOAD PERIODIC CURRENT COUNTER
               MOV AX,WORD PTR [BX + SI + PERIOD_RELOAD]
               MOV WORD PTR [BX + SI + PERIOD_CURRENT],AX
```

```
; THEN PLACE THAT TASK IN THE READY QUEUE

PUT_IN_Q0:      INC WORD PTR [TOP_OF_Q]
                MOV SI, WORD PTR [TOP_OF_Q]
                MOV AL,CL
                DEC AL ; AL NOW CONTAINS THE TASK NUMBER
                MOV BYTE PTR [SI],AL ; PLACE TASK ON READY QUEUE AND
                MOV BYTE PTR [NEW_TASK_IN_Q],1 ; INDICATE THAT A NEW TASK WAS PLACED IN Q

NEXT_CHECK0:
                LOOP CHECK0 ; REPEAT FOR THE NEXT TASK

; NOW CHECK TASKS AGAIN FOR ANY NORMAL TIME OUTS
                MOV CX,NUM_OF_TASKS
                MOV BX,OFFSET TASK_PARAM
                MOV DL,PARAM_SIZE
CHECK1:         MOV AL,CL
                DEC AL ; AL NOW CONTAINS THE TASK NUMBER
                MUL DL
                MOV SI,AX ; SI CONTAINS OFFSET IN TASK RECORD
                CMP WORD PTR [BX + SI + TIME_OUT],0
                JZ NEXT_CHECK1 ; SKIP IF TASK IS NOT WAITING FOR ANY TIMEOUT
                DEC WORD PTR [BX + SI + TIME_OUT] ; DECREMENT TIMEOUT
                JNZ NEXT_CHECK1
; IF TIME OUT FINISHED, THEN PLACE THAT TASK IN THE READY QUEUE

PUT_IN_Q: INC WORD PTR [TOP_OF_Q]
                MOV SI, WORD PTR [TOP_OF_Q]
                MOV AL,CL
                DEC AL ; AL NOW CONTAINS THE TASK NUMBER
                MOV BYTE PTR [SI],AL ; PLACE TASK ON READY QUEUE AND
                MOV BYTE PTR [NEW_TASK_IN_Q],1 ; INDICATE THAT A NEW TASK WAS PLACED IN Q

NEXT_CHECK1:
                LOOP CHECK1 ; CHECK THE NEXT TASK

;CHECK IF THERE IS A NEED FOR A TASK CHANGE
                CMP BYTE PTR [NEW_TASK_IN_Q],1
                JNE CARRY_ON
                CMP BYTE PTR [RUNNING], IDLE_TASK
                JNE CARRY_ON ; RTOS CAN ONLY INTERRUPT THE IDLE TASK

; A TASK CHANGE IS REQUIRED
                MOV BYTE PTR [NEW_TASK_IN_Q],0 ; CLEAR NEW TASK IN Q FLAG
; NOW SAVE STACK POINTER FOR CURRENT TASK
                MOV BX,OFFSET TASK_PARAM
                MOV AL,BYTE PTR [RUNNING] ; GET CURRENT TASK NUMBER
                MOV DL,PARAM_SIZE
                MUL DL ; AX = AL * DL
                MOV SI,AX ; SI CONTAINS OFFSET IN TASK RECORD
                MOV WORD PTR [BX + SI + SP_STORE],SP ; SAVE SP FOR THIS TASK

                ; NOW INITIATE A TASK CHANGE
                ; GET THE NEXT TASK NUMBER WHICH IS READY TO RUN
                CALL SHIFT_READYQ
```

```
                ; NOW RESTORE STACK POINTER FOR THE NEW TASK
                MOV BX,OFFSET TASK_PARAM
                MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS NEW TASK NUMBER
                MOV DL,PARAM_SIZE
                MUL DL ; AX = AL * DL
                MOV SI,AX ; SI CONTAINS OFFSET IN TASK RECORD
                MOV SP,WORD PTR [BX + SI + SP_STORE] ; GET SP FOR THIS TASK
CARRY_ON:
                POP_INT_REGS
                IRET ; CONTINUES WITH A NEW TASK IF TASK CHANGED
OS_TICK_TIMER ENDP
;============================================================
;============================================================

; SHIFT READY QUEUE DOWN ONE PLACE
SHIFT_READYQ PROC NEAR
                PUSH DI
                PUSH SI
                PUSH DX
                PUSH ES
                PUSH AX
                MOV DI,OFFSET RUNNING
                MOV SI,OFFSET RUNNING + 1
                MOV DX,WORD PTR [TOP_OF_Q] ; GET POINTER TO TOP_OF_Q
                ADD DX,2 ; AND POINT 2 BYTES UP
                MOV AX,WORD PTR [MY_CODE_SEG]
                MOV ES,AX
                CLD     ; MOVSB WILL INCREMENT SI AND DI AUTOMATICALLY
SHIFT:          MOVSB   ; MOVE BYTE FROM LOCATION DS:SI INTO LOCATION ES:DI
                        ; INC SI and INC DI done automatically in above instruction
                CMP SI,DX
                JNE SHIFT
                MOV SI, WORD PTR [TOP_OF_Q]
                MOV DI, OFFSET RUNNING
                CMP DI,SI
                JE LIMIT ; TOP_OF_Q CAN NEVER GO BELOW 'RUNNING'
                DEC WORD PTR [TOP_OF_Q]
LIMIT: POP AX
                POP ES
                POP DX
                POP SI
                POP DI
                RET
SHIFT_READYQ ENDP

;============================================================
;============================================================

; CREATE A TASK
; ON ENTRY:
;               DX POINTS TO START OF TASK
;               AL CONTAINS THE TASK NUMBER
;
; SET UP THE TASK ADDRESS ON STACK AND
```

```
; PLACE IT IN READY QUEUE

CREATE_TASK PROC NEAR
               PUSH_ALL_REGS
; PUT TASK NUMBER IN READY QUEUE
               INC WORD PTR [TOP_OF_Q] ; TOP_OF_Q IS A POINTER
               MOV SI, WORD PTR [TOP_OF_Q]
               MOV BYTE PTR [SI],AL
               MOV BYTE PTR [NEW_TASK_IN_Q],1

; FIND THE STACK FOR THE TASK NUMBER HELD IN AL
               MOV BX,OFFSET TASK_PARAM
               MOV CL,PARAM_SIZE
               MUL CL ; AX = AL*CL
               MOV SI,AX

; TASK ADDRESS HAS NOW TO BE SAVED ON THE TASK STACK
; SINCE ALL THE PROGRAM RESIDES ON ONE SEGMENT, ALL CALLS ARE NORMALLY NEAR
; HERE WE SIMULATE PUSHF, PUSH CS, PUSH IP CARRIED OUT BY THE INTERRUPT

               MOV WORD PTR [BX + SI + STACK_AREA - 1],0
                       ; STORE FLAGS OF TASK, INITIALLY ALL ZERO (LOW BYTE FIRST)
               MOV WORD PTR [BX + SI + STACK_AREA - 3],CS
                       ; STORE CS OF TASK (LOW BYTE FIRST)
               MOV WORD PTR [BX + SI + STACK_AREA - 5],DX
                       ; STORE IP OF TASK (LOW BYTE FIRST)
               MOV CX,BX
               ADD CX,SI
               ADD CX,(STACK_AREA - 2*REG_PUSHES - 5) ; CX NOW CONTAINS CORRECT SP OFFSET
               ; READY FOR POP_ALL_REGS AND IRET IN TIMER ROUTINE
               MOV WORD PTR [BX + SI + SP_STORE],CX ; SAVE SP FOR THIS TASK

               POP_ALL_REGS
               RET
CREATE_TASK ENDP
;==============================================================
;==============================================================

; SET TASK TO EXECUTE PERIODICALLY, EVERY CERTAIN NUMBER OF TICKS
; ON ENTRY:
;       DX CONTAINS THE NUMBER OF TICKS TO WAIT FOR
; NO TASK CHANGE

PERIODIC PROC NEAR
               PUSH_ALL_REGS
               MOV BX,OFFSET TASK_PARAM
               MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS CURRENT TASK NUMBER
               MOV CL,PARAM_SIZE
               MUL CL ; AX=AL*CL
               MOV SI,AX
               MOV WORD PTR [BX + SI + PERIOD_CURRENT],DX

               MOV WORD PTR [BX + SI + PERIOD_RELOAD],DX
               ; TIMEOUT DATA NOW SAVED ON TASK PARAMETER STORAGE
               POP_ALL_REGS
               RET
```

```
PERIODIC ENDP

;===========================================================
;===========================================================

; PLACE A TASK IN A WAIT STATE, JUST WAITING FOR THE
; PREVIOUSLY ASSIGNED PERIODIC INTERVAL TO PASS

WAIT_PERIOD PROC NEAR
               PUSH_INT_REGS
               ; SAVE STACK POINTER FOR CURRENT TASK
               MOV BX,OFFSET TASK_PARAM
               MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS NEW TASK NUMBER
               MOV CL,PARAM_SIZE
               MUL CL
               MOV SI,AX
               MOV WORD PTR [BX + SI + SP_STORE],SP ; SAVE SP FOR THIS TASK

               ; NOW INITIATE A TASK CHANGE
               ; SHIFT DOWN READY QUEUE BY ONE, AND
               ; GET NEXT TASK NUMBER WHICH IS READY TO RUN
               CALL SHIFT_READYQ

               ; NOW RESTORE STACK POINTER FOR THE NEW TASK
               MOV BX,OFFSET TASK_PARAM
               MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS NEW TASK NUMBER
               MOV CL,PARAM_SIZE
               MUL CL
               MOV SI,AX
```

```
                MOV SP,WORD PTR [BX + SI + SP_STORE] ; GET SP FOR THIS TASK
                POP_INT_REGS
                IRET ; START NEW TASK WITH THIS IRET INSTRUCTION
WAIT_PERIOD ENDP

;============================================================
;============================================================

; PLACE A TASK IN A WAIT STATE, JUST WAITING FOR A CERTAIN NUMBER OF TICKS
; ON ENTRY:
;               DX CONTAINS THE NUMBER OF TICKS TO WAIT FOR
;               CS:IP ALREADY PUSHED ON STACK BY THE 'CALL OS_WAITT'

WAITT PROC NEAR
                PUSH_INT_REGS
                MOV BX,OFFSET TASK_PARAM
                MOV AL,BYTE PTR [RUNNING]     ; AL NOW CONTAINS CURRENT TASK NUMBER
                MOV CL,PARAM_SIZE
                MUL CL                        ; AX=AL*CL
                MOV SI,AX
                MOV WORD PTR [BX + SI + TIME_OUT],DX
                ; TIMEOUT DATA NOW SAVED ON TASK PARAMETER STORAGE

                ; NOW SAVE STACK POINTER FOR CURRENT TASK
                MOV WORD PTR [BX + SI + SP_STORE],SP ; SAVE SP FOR THIS TASK

                ; NOW INITIATE A TASK CHANGE
                ; SHIFT DOWN READY QUEUE BY ONE, AND
                ; GET NEXT TASK NUMBER WHICH IS READY TO RUN
                CALL SHIFT_READYQ

                ; NOW RESTORE STACK POINTER FOR THE NEW TASK
                MOV BX,OFFSET TASK_PARAM
                MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS NEW TASK NUMBER
                MOV CL,PARAM_SIZE
                MUL CL          ; AX=AL*CL
                MOV SI,AX
                MOV SP,WORD PTR [BX + SI + SP_STORE] ; GET SP FOR THIS TASK
                POP_INT_REGS
                IRET ; START NEW TASK WITH THIS IRET INSTRUCTION
WAITT ENDP

;============================================================
;============================================================
WAITI PROC NEAR
                RET
WAITI ENDP

;============================================================
;============================================================
; PLACE A TASK IN A WAIT STATE, JUST WAITING FOR A SIGNAL
; IF SIGNAL ALREADY PRESENT, CLEAR SIGNAL AND CONTINUE
; OTHER WIASE MAKE A TASK CHANGE
; ON ENTRY:
;               DX CONTAINS THE NUMBER OF TICKS TO WAIT FOR
;               CS:IP ALREADY PUSHED ON STACK BY THE 'CALL OS_WAITT'
```

```
WAITS PROC NEAR ; STAY WAITING FOR A SIGNAL - TASK CHANGE
               PUSH_INT_REGS
               MOV BX,OFFSET TASK_PARAM
               MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS CURRENT TASK NUMBER
               MOV CL,PARAM_SIZE
               MUL CL ; AX=AL*CL
               MOV SI,AX
               CMP BYTE PTR [BX + SI + SIGNAL_FLAG],1
               JE CONT1 ; SIGNAL ALREADY PRESENT, HENCE CONTINUE

               ; IF NOT, THEN SET FLAG AND MAKE A TASK CHANGE
               MOV BYTE PTR [BX + SI + SIGNAL_FLAG],1
               ; NOW SAVE STACK POINTER FOR CURRENT TASK
               MOV WORD PTR [BX + SI + SP_STORE],SP ; SAVE SP FOR THIS TASK

               ; NOW INITIATE A TASK CHANGE
               ; SHIFT DOWN READY QUEUE BY ONE, AND
               ; GET NEXT TASK NUMBER WHICH IS READY TO RUN
               CALL SHIFT_READYQ

               ; NOW RESTORE STACK POINTER FOR THE NEW TASK
               MOV BX,OFFSET TASK_PARAM
               MOV AL,BYTE PTR [RUNNING] ; AL NOW CONTAINS NEW TASK NUMBER
               MOV CL,PARAM_SIZE
               MUL CL ; AX=AL*CL
               MOV SI,AX
               MOV SP,WORD PTR [BX + SI + SP_STORE] ; GET SP FOR THIS TASK
               JMP EXIT1

CONT1:         MOV BYTE PTR [BX + SI + SIGNAL_FLAG],0
EXIT1:         POP_INT_REGS
               IRET ; START NEW TASK WITH THIS IRET INSTRUCTION, IF REQUIRED
WAITS ENDP

;============================================================
;============================================================

; SEND A SIGNAL TO TASK.
; IF TASK WAS ALREADY WAITING FOR THE SIGNAL, THEN THAT TASK
; IS PLACED IN THE READY QUEUE.
; ON ENTRY:
;              AL CONTAINS THE TASK NUMBER TO BE SIGNALLED

SIGNAL_TASK PROC NEAR  ; SEND A SIGNAL TO A TASK SO THAT IT CAN RESUME
                       ; NO TASK CHANGE IS MADE
       PUSH_ALL_REGS
       PUSH AX         ; SAVE TASK NUMBER
       MOV BX,OFFSET TASK_PARAM
       MOV CL,PARAM_SIZE
       MUL CL ; AX=AL*CL
       MOV SI,AX
       CMP BYTE PTR [BX + SI + SIGNAL_FLAG],0
       POP AX ; GET TASK NUMBER AGAIN
       JE TASK2WAIT ; TASK WAS NOT WAITING, HENCE JUST SET FLAG

       ; TASK WAS ALREADY WAITING, HENCE PUT IT IN READY QUEUE
```

```
        MOV BYTE PTR [BX + SI + SIGNAL_FLAG],0
        INC WORD PTR [TOP_OF_Q] ; TOP_OF_Q IS A POINTER
        MOV SI, WORD PTR [TOP_OF_Q]
        MOV BYTE PTR [SI],AL
        MOV BYTE PTR [NEW_TASK_IN_Q],1
        JMP EXIT2

TASK2WAIT:      MOV BYTE PTR [BX + SI + SIGNAL_FLAG],1
EXIT2:          POP_ALL_REGS
                RET
SIGNAL_TASK ENDP

;============================================================
;============================================================

;========================================================
; OSTICKFT.INC
;
;
;       Written by Paul P. Debono ( JAN 2012 )
;
;========================================================

; RTOS DATA AREA
;

;
MY_CODE_SEG     DW 0 ; STORAGE FOR THE CODE SEGMENT REGISTER
```

```
; The OldTick variable holds the old value of the low word
; (DX) returned by INT 1AH to get the system time in ticks
; since midnight in CX:DX

OldTick        DW 0

NEW_TASK_IN_Q  DB 0
TOP_OF_Q       DW OFFSET RUNNING ; pointer to the top of the ready queue
RUNNING        DB IDLE_TASK ; contains the task number of the currently running task
READY_Q        DB (NUM_OF_TASKS + 2) DUP (IDLE_TASK)
;
;
; STRUCTURE DEFINITIONS, UNFORTUNATELY NOT SUPPORTED BY EMU8086
; 50 BYTES (PARAM_SIZE) PER TASK
;
;TASK_PARAM STRUC
;       SIGNAL_FLAG DB 0
;       INT_NUM DB 0
;       SP_STORE DW 0
;        TIME_OUT DW 0
;       PERIOD_CURRENT DW 0
;       PERIOD_RELOAD DW 0
;       STACK_AREA DB 40 DUP (0)
;TASK_PARAM ENDS

TASK_PARAM DB PARAM_SIZE * (NUM_OF_TASKS + 1) DUP (0)
;==================================================
```

Appendix H 8051 Instruction Set

8051 Alphabetical List of the Instruction Set

- ACALL Absolute Call
- ADD, ADDC Add Accumulator (With Carry)
- AJMP Absolute Jump
- ANL Bitwise AND
- CJNE Compare and Jump if Not Equal
- CLR Clear Register or Bit
- CPL Complement Register or Bit
- DA Decimal Adjust
- DEC Decrement Register
- DIV Divide Accumulator by B
- DJNZ Decrement Register and Jump if Not Zero
- INC Increment Register
- JB Jump if Bit Set
- JBC Jump if Bit Set and Clear Bit
- JC Jump if Carry Set
- JMP Jump to Address
- JNB Jump if Bit Not Set
- JNC Jump if Carry Not Set
- JNZ Jump if Accumulator Not Zero
- JZ Jump if Accumulator Zero
- LCALL Long Call
- LJMP Long Jump
- MOV Move Memory
- MOVC Move Code Memory
- MOVX Move Extended Memory
- MUL Multiply Accumulator by B
- NOP No Operation
- ORL Bitwise OR
- POP Pop Value From Stack
- PUSH Push Value Onto Stack
- RET Return From Subroutine
- RETI Return From Interrupt
- RL Rotate Accumulator Left
- RLC Rotate Accumulator Left Through Carry
- RR Rotate Accumulator Right

- RRC Rotate Accumulator Right Through Carry
- SETB Set Bit
- SJMP Short Jump
- SUBB Subtract From Accumulator With Borrow
- SWAP Swap Accumulator Nibbles
- XCH Exchange Bytes
- XCHD Exchange Digits
- XRL Bitwise Exclusive OR

Bibliography

1. AYALA, K.J., 1999. *8051 Microcontroller: Architecture, Programming, and Applications.* 2nd edn. Delmar Thomson Learning.
2. BARNETT, R.H., 1994. *The 8051 Family of Microcontrollers.* 1st edn. Upper Saddle River, NJ, USA: Prentice Hall PTR.
3. CALCUTT, D.M., COWAN, F.J. and PARCHIZADEH, G.H., 1998. *8051 Microcontrollers Hardware, Software and Applications.* London, UK: Arnold.
4. CHEW, M.T. and GUPTA, G.S., 2005. *Embedded Programming with Field-Programmable Mixed-Signal Microcontrollers.* Silicon Laboratories.
5. HUANG, H., 2009. *Embedded System Design with the C8051.* Stanford, CT, USA: Cengage Learning.
6. LABROSSE, J.J., 2002. *Embedded Systems Building Blocks.* 2nd edn. San Francisco, CA, USA: CMP Books.
7. LABROSSE, J.J., 2002. *MicroC/OS-II, The Real-Time Kernel.* 2nd edn. San Francisco, CA, USA: CMP Books.
8. MACKENZIE, I.S., 1998. *The 8051 Microcontroller.* 3rd edn. Upper Saddle River, NJ, USA: Prentice Hall PTR.
9. MAZIDI, M.A. and MAZIDI, J.G., 1999. *The 8051 Microcontroller and Embedded Systems with Disk.* 1st edn. Upper Saddle River, NJ, USA: Prentice Hall PTR.
10. PONT, M.J., 2002. *Embedded C.* 1st edn. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
11. PONT, M.J., 2001. *Patterns for time-triggered embedded systems: building reliable applications with the 8051 family of microcontrollers.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.
12. PREDKO, M., 1999. *Programming and Customizing the 8051 Microcontroller.* New York, NY, USA: McGraw-Hill, Inc.
13. SCHULTZ, T.W., 2004. *C And The 8051.* Pagefree Publishing.
14. SCHULTZ, T.W., 1999. *C and the 8051 (volume II): building efficient applications.* Upper Saddle River, NJ, USA: Prentice Hall PTR.
15. SCHULTZ, T.W., 1997. *C and the 8051: Hardware, Modular Programming and Multitasking with Cdrom.* 2nd edn. Upper Saddle River, NJ, USA: Prentice Hall PTR.
16. STEWART, J.W., 1999. *The 8051 microcontroller (2nd ed.): hardware, software and interfacing.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.
17. THORNE, M., 1986. *Programming the 8086/8088 for the IBM PC and compatibles.* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc.
18. VARIOUS, 1993. *MCS51 Microcontroller Family User's Manual.* Santa Clara, CA, USA: Intel Corporation.

19. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multi-programming in a Hard Real-Time Environment,"J. ACM, vol. 20, no. 1, pp. 40–61, 1973.
20. J. Blaut, 2004, "8051 RTOS", B.Sc. Eng. Thesis, University of Malta.

Index for Part I

Index for Part II

End Notes

1. The original idea for this RTOS came from the book "C and the 8051 – Building Efficient Applications – Volume II" by Thomas W. Schultz and published by Prectice Hall (0-13-521121-2). In this book, Prof. Schultz discusses the development of two real-time kernels. The first one is the RTKS which I corrected and developed into PaulOS co-operative RTOS. The second one is the RTKB which I also corrected, modified and developed into MagnOS pre-emptive RTOS. Both operating systems, RTKS and RTKB as written in the book are not fully functional, contain some errors and lack some essential components. I did correspond with Prof. Schultz and sent him my modifications and final versions of the programs which he later acknowledged in the 3rd edition of the book "C and the 8051", again published by Prentice-Hall (0-58961-237-X). So I am particularly grateful to Prof. Schultz for being the catalyst of my increased interest in RTOSs.
2. The development of a pre-emptive RTOS, named RTKB is described in the book "C and the 8051 – Building Efficient Applications – Volume II" published by Prentice-Hall (0-13-521121-2. A third edition was later published having the ISBN 1-58961-237-X where the author acknowledged my contribution to the development of a working version of his original RTOS.